

# Denotational Recurrence Extraction for Amortized Analysis

JOSEPH W. CUTLER, Wesleyan University

DANIEL R. LICATA, Wesleyan University

NORMAN DANNER, Wesleyan University

A typical way of analyzing the time complexity of functional programs is to extract a recurrence expressing the running time of the program in terms of the size of its input, and then to solve the recurrence to obtain a big- $O$  bound. For recurrence extraction to be compositional, it is also necessary to extract recurrences for the size of outputs of helper functions. Previous work has developed techniques for using logical relations to state a formal correctness theorem for a general recurrence extraction translation: a program is bounded by a recurrence when the operational cost is bounded by the extracted cost, and the output value is bounded, according to a value bounding relation defined by induction on types, by the extracted size. This previous work supports higher-order functions by viewing recurrences as programs in a lambda-calculus, or as mathematical entities in a denotational semantics thereof. In this paper, we extend these techniques to support amortized analysis, where costs are rearranged from one portion of a program to another to achieve more precise bounds. We give an intermediate language in which programs can be annotated according to the banker's method of amortized analysis; this language has an affine type system to ensure credits are not spent more than once. We give a recurrence extraction translation of this language into a recurrence language, a simply-typed lambda-calculus with a cost type, and state and prove a bounding logical relation expressing the correctness of this translation. The recurrence language has a denotational semantics in preorders, and we use this semantics to solve recurrences, e.g. analyzing binary counters and splay trees.

## ACM Reference Format:

Joseph W. Cutler, Daniel R. Licata, and Norman Danner. 2020. Denotational Recurrence Extraction for Amortized Analysis. 1, 1 (May 2020), 48 pages. <https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

## 1 INTRODUCTION

An important aspect of programming is predicting how much of certain resources, such as time or space, a program will require to execute. A very common technique for analyzing time complexity of functional programs is the *extract-and-solve* method, in which one extracts a recurrence expressing an upper bound on the cost of the program in terms of the size of its input, and then solves the recurrence to obtain a big- $O$  bound. Typically, the connection between the original program and the extracted recurrence is left informal, relying on an intuitive understanding that the extracted recurrence correctly models the program. Previous work [9, 10, 20, 22] has begun to explore more formal techniques for relating programs and extracted recurrences. The process of extracting a recurrence is modeled by a monadic translation into the writer monad (translating a program to also “output” its cost), combined with an abstraction of values as sizes (e.g. abstracting a list as its length). A *bounding logical relation* relates a program to its cost, and more generally relates

---

Authors' addresses: Joseph W. Cutler, Wesleyan University, Middletown, Connecticut, [jwcutler@wesleyan.edu](mailto:jwcutler@wesleyan.edu); Daniel R. Licata, Wesleyan University, Middletown, Connecticut, [dlicata@wesleyan.edu](mailto:dlicata@wesleyan.edu); Norman Danner, Wesleyan University, Middletown, Connecticut, [ndanner@wesleyan.edu](mailto:ndanner@wesleyan.edu).

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2020 Association for Computing Machinery.

XXXX-XXXX/2020/5-ART \$15.00

<https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

values to a notion of *size* according to a value bounding relation defined by induction on types. For example, the size of a list value can be its length, or its length paired with the size of its maximum element, as necessary for a given analysis, while at function type, the size is itself a recurrence, so the technique provides higher-order recurrences for higher-order functions. The fundamental *bounding theorem* states that a program and its extraction are related by the bounding relation, which in particular implies that its actual runtime cost is bounded by the extracted prediction. Finally, *denotational models* of the recurrences are used to abstract values as sizes and obtain big- $O$  bounds. Previous work develops this methodology for functional programs with numbers and lists [10], inductive types with structural recursion [9], and general recursion [22].

However, one kind of analysis not covered by this previous work is *amortized analysis*, where the cost of expensive operations is redistributed to less expensive ones, yielding a more precise bound for a collection of operations. A standard motivating example of amortized analysis is implementing a binary counter using a list of bits, with an operation `set : nat -> [bit]` for setting the counter to a particular number, implemented using a helper function `inc : [bit] -> [bit]` for incrementing the counter by one:

```
inc [] = [1]                set 0 = []
inc (0::bs) = 1::bs         set n = inc (set (n-1))
inc (1::bs) = 0::(inc bs)
```

For simplicity, we define the cost to be the number of times a bit is flipped (0 to 1 or 1 to 0). Following the usual extract-and-solve method, we might obtain worst-case recurrences  $T_{\text{inc}}(l) = 1 + T_{\text{inc}}(l-1)$  (where  $l$  is the length of the input list) and  $T_{\text{set}}(n) = T_{\text{set}}(n-1) + T_{\text{inc}}(n)$  (where  $n$  is the number the counter is set to), so  $T_{\text{inc}} \in O(n)$  and  $T_{\text{set}} \in O(n^2)$ . This analysis requires bounding the length of the input list to `inc` by  $n$ , while a more precise analysis might observe that this length is at most  $O(\log n)$ , and obtain  $O(n \log n)$  for `set`. However, both of these bounds are imprecise: in fact,  $T_{\text{set}} \in O(n)$  overall, intuitively because `inc` is constant time when the first bit is zero, and the increments  $0 \rightarrow 1 \rightarrow 2 \rightarrow \dots \rightarrow n$  execute this case often enough.

This more precise bound can be obtained using the *banker's method* for amortized analysis [37], in which one can “prepay” time cost to generate “credits” that are “spent” later to reduce time cost, rearranging the accounting of costs from one portion of a program to another (in particular, generating a credit costs 1 unit of time, while spending a credit reduces the cost by 1 unit of time). In this example, we maintain the invariant that one credit is attached to every 1 bit in the counter representation. The *amortized cost* of flipping a bit from 0 to 1 is then 2 units of time—one for the actual bit flip plus one to generate the credit. However, the amortized cost of flipping a bit from 1 to 0 is 0 units of time—the bit flip takes one unit of time, but that is paid for by the credit. Using these new amortized costs, we can see that  $T_{\text{inc}}(n)$  is  $O(1)$  amortized: in the case where the first bit is 0, we flip it to 1, which costs 2 units of time, and stop. In the case where the first bit is 1, we flip it *for free* to 0, and then make a recursive call, which inductively is bounded by 2. So  $T_{\text{inc}}(n) = 2$ , which means that  $T_{\text{set}}(n) = 2n$ , amortized. Since a single run of `set` starts with no credits, its actual cost will be bounded by the amortized cost  $2n$ : all of the credits spent during the call to `set`, which subtract from the cost, must have been created earlier, incurring a cost which balances out the gain garnered from spending it.

In this paper, we extend the formal approach to recurrence extraction (most directly following [9, 10, 20]) to the accounting/banker's method for amortized analysis. This requires us to move from a relatively standard source language based on the simply-typed  $\lambda$ -calculus with inductive datatypes to a more specialized one. We do not expect amortization policies (e.g. generate a credit when flipping a bit from 0 to 1, to be spent when flipping a bit from 1 to 0) to be automatically inferable in the general case—these policies are the part of an amortized analysis that requires the most cleverness. To notate these policies, we use an *intermediate language*  $\lambda^A$  (Section 2),

which has “effectful” operations for generating and spending credits (create and spend), as well as a modal type operator  $!_\ell$  for associating credits with values (e.g. storing a credit with each 1 in a bit list). The type  $!_\ell A$  classifies a value of type  $A$  that has  $\ell$  credits associated with it. To correctly manage credits, this intermediate language is based on a form of linear logic, which prevents spending the same credit more than once; in particular,  $\lambda^A$  is an affine lambda calculus with all of the standard connectives  $\otimes, \oplus, \&, \multimap, !$  plus multiplicities  $!^k A$  (where  $k$  is a positive number) for tracking multiple-use values. The type structure of the intermediate language is inspired by the credits (written as  $\diamond$ ) of [17, 18],  $n$ -linear types (e.g. [5, 13, 27, 31]), and the uses of credits and linear logic in automatic amortized resource analysis (AARA) (e.g. [14, 19, 24]). However, our goal for the intermediate language is different than the above works: we formalize an extract-and-solve-a-recurrence technique for amortized analysis.

To this end, we give a translation of the intermediate language  $\lambda^A$  into a *recurrence language*  $\lambda^C$ . The recurrence language, following [9, 20], is a standard simply-typed  $\lambda$ -calculus with a base type for costs (linearity is not needed at this stage). It is equipped with an inequality judgment  $E \leq_T E'$  that can be used to express upper bounds. The translation we define here extracts a recurrence for the *amortized* cost of the program (where the costs have been “rearranged”), by translating the credit generation and spending operations in  $\lambda^A$  to modifications of the cost. We define a bounding relation (a cross-language logical relation) for the amortized case, and prove that a term is related to its extraction. As a corollary, we obtain that the amortized cost of running a program from  $\lambda^A$  is bounded by the cost component of its translation into  $\lambda^C$ ; for programs that use no external credits, this gives a bound on its actual cost as well. The recurrence language, recurrence extraction and bounding theorem are described in Section 3. Next, we use a denotational semantics of the recurrence language in preorders, similar to [9], to justify the consistency of the recurrence language  $\leq$  judgment, and to simplify and solve extracted recurrences (Section 4).

The version of  $\lambda^A$  and the recurrence extraction presented through Section 4 allows a statically fixed number of credits to be stored with each element of a data structure (e.g. 1 credit on element of a list, so  $n$  credits overall). For some analyses, it is necessary to choose the number of credits stored with an element dynamically; for example, when implementing balanced binary search trees via splay trees [35], the number of credits stored at each node in the tree is a function of the size of the subtree rooted at that node, which varies for different tree nodes. To support such analyses, in Section 5, we extend  $\lambda^A$  with existential quantifiers over credit variables, and use them to code a portion of [29]’s analysis of splay trees in our system.

While our recurrence extraction and the denotational semantics in preorders are given as automatic language-to-language translations, there are two phases of the analysis that, for this paper, require manual intervention. On the front-end, a source program written in a standard functional language must be annotated with its credit usage policy by translating it into  $\lambda^A$ , and on the back-end the extracted recurrence must be simplified and solved. We diagram the overall process of extracting and solving a recurrence in Figure 1, where the first and last steps are manual and the middle two are automatic. While further automation is a worthwhile goal, and there are interesting questions about how to automate the annotation and solution steps, our main motivation in this paper is to formally justify the extract-and-solve method for amortized analysis, a technique that we teach and that is typically used by practitioners. Connecting the extracted recurrence in terms of user-defined notions of size to the operational cost is the least justified step in this process, and so a formal account of it has important foundational value. It could likewise have important practical value: because students and practitioners are trained in the use of cost recurrences, reverse-engineering a recurrence that yields a worse-than-expected cost bound to the (mis)implementation may require a lower cognitive load than doing the same with more

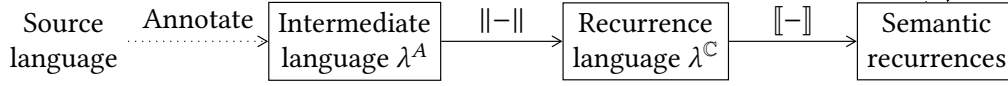


Fig. 1. Recurrence Extraction Pipeline

sophisticated techniques. Moreover, though this technique is less automated than others, it can handle at least some examples that existing techniques cannot—to our knowledge, splay trees cannot be analyzed by the existing automatic techniques. We give a detailed comparison with related work in Section 6.

## 2 INTERMEDIATE LANGUAGE $\lambda^A$

In this section we discuss the static and operational semantics of  $\lambda^A$ , which is an *affine* lambda calculus—it permits weakening (unused variables) but not contraction (duplication of variables). It includes some standard connectives of linear logic, such as positive/eager/multiplicative products ( $\otimes$  and  $1$ ), sums/coproducts ( $\oplus$ ), and functions ( $\multimap$ ), as well as negative/lazy/additive products ( $\&$ ). The language has two basic datatypes, natural numbers ( $\mathbb{N}$ ) and (eager) lists ( $\text{List}(A)$ ), both with structural recursion (though we expect these techniques to extend to all strictly positive inductive types [8, 9]).

In addition to these,  $\lambda^A$  contains some constructs specific to its role as an intermediate language for expressing amortized analyses. First, instead of fixing the operational costs of  $\lambda^A$ 's programs themselves, we include a tick operation which costs 1 unit of time, and assume that the translation of a program into  $\lambda^A$  has annotated the program with sufficient ticks to model the desired operational cost [7] (for example, we can charge only for bit flips in the above binary counter program).

Second, we have operations *create* and *spend* for creating and spending credits, which respectively increase and decrease the *amortized* cost of the program *without changing* the true operational cost.

Third, we have a type constructor  $!_\ell A$ , where a value of this type is a value of type  $A$  with  $\ell$  credits attached; its introduction and elimination rules allow for the movement of credits around a program. The combination of *spend* and the  $!_\ell$  modality motivates our affine type system: because spending credits decreases the amortized cost of a program, we must ensure that a credit is spent only once, so credits should not be duplicated; because credits can be stored in values, values cannot in general be duplicated as well. However,  $\lambda^A$  does allow credit weakening—choosing not to spend available credits—because this increases the amortized cost (relative to spending the credits), and we are interested in upper bounds on running time. While the basic affine type system allows a variable to be used only once, to simplify the expression of programs that use a variable a fixed number of times, we use  $n$ -linear types (see e.g. [5, 13, 27, 31]), where variables are annotated with a multiplicity  $k$ , and can be used at most  $k$  times.<sup>1</sup> This is internalized by a modality  $!^k A$ , which represents an  $A$  that can be used at most  $k$  times. We additionally allow  $k$  to be  $\infty$ , in which case  $!^\infty A$  is the usual exponential of linear logic, allowing unrestricted use. Using this modality, standard functional programs can be coded in  $\lambda^A$ , but our current recurrence extraction does not handle the  $!^\infty$  fragment very well, as explained below—at present, we use  $!^\infty$  mainly as a technical device for typing recursors. It is technically convenient to combine the two modalities into one type former  $!_\ell^k A$ , which represents an  $A$  that can be used  $k$  times, which also has  $\ell$  credits attached (total, not  $\ell$  credits with each use). Because  $k$  is a coefficient but  $\ell$  is an additive constant, the individual

<sup>1</sup>While Girard's notation for multiplicities is  $!_k A$  [13], we write superscripts following [5], and write subscripts for the credit-storing modality, which is used more frequently in our system.

modalities are recovered as  $!^k A := !_0^k A$  and  $!_\ell A := !_\ell^1 A$ . In pure affine logic, one can think of  $!^k A$  as  $X \otimes \dots \otimes X \otimes A \otimes \dots \otimes A$  with  $\ell$   $X$ s and  $k$   $A$ 's (in the case where  $k$  and  $\ell$  are finite), for an atomic proposition  $X$  representing a single credit. However, our judgmental presentation is easier to work with for our bounding relation and theorem below, and the  $n$ -linear modality  $!^k A$  ensures that additional invariant that it is the *same* value that can be used  $k$  times, i.e. it only allows the diagonal of  $A \otimes \dots \otimes A$ .

## 2.1 Type System

In Fig. 2 we define a typing judgment of the form  $\Gamma \vdash_f M : A$ , where  $\Gamma$  is a standard context  $x_1 : A_1, x_2 : A_2, \dots, x_n : A_n$  and  $f$  is a *resource* term of the form  $a_1 x_1 + a_2 x_2 + \dots + a_n x_n + \ell$ , where  $x_1, \dots, x_n$  are the variables in  $\Gamma$  and  $a_i$  and  $\ell$  are natural numbers or  $\infty$ . The resource term  $f$  can be thought of as annotating each variable  $x_i$  with the number of times  $a_i$  that it is allowed to occur, and additionally annotating the judgment with a nonnegative “bank”  $\ell$  of available credits that can be used. For example, the judgment  $x : A, y : B, z : C \vdash_{3x+2y+0z+2} M : D$ , means that  $M$  is a term of type  $D$ , which may use  $x$  at most 3 times,  $y$  at most twice,  $z$  not at all, and has access to 2 credits. We consider these resource terms up to the usual arithmetic identities (associativity, unit, commutativity, distributivity,  $0f = 0$ ,  $\infty k = \infty$  otherwise, etc.). In the admissible substitution rule, we write  $g[f/x]$  to denote the result of normalizing the textual substitution of  $f$  for  $x$  in  $g$  according to these identities; e.g.  $(3x + 2y + 2)[10a + 11b + 3/x] = 30a + 33b + 2y + 11$ . Our judgmental presentation of  $n$ -linear types differs from some existing ones— the reader more familiar with Girard [13] may read  $\Gamma \vdash_f M : A$  as analogous to  $!_f \Gamma \vdash M : A$ — but this type system was derived as an instance of a general framework for modal types [26], which, for our purposes, simplifies the presentation of standard metatheorems like substitution. Note that the resource terms  $f$  play a different role than the resource polynomials in AARA [14], which provide a mechanism for measuring the size and credit allocation in a data structure. The resource terms are also affine in the sense of a polynomial—the exponent of every variable is 1, except for the constant term  $\ell$ —but we will avoid this meaning of affine to avoid confusion with “affine logic” (allowing weakening but not contraction).

**2.1.1 Structural Rules.** The rules make three structural principles admissible:

**THEOREM 2.1 (ADMISSIBLE STRUCTURAL RULES).**

- *Resource Weakening:* Write  $g \geq f$  for the coefficient-wise partial order on resource terms ( $a_1 x_1 + a_2 x_2 + \dots + \ell \geq b_1 x_1 + b_2 x_2 + \dots + \ell'$  iff  $a_i \geq b_i$  for all  $i$  and  $\ell \geq \ell'$ ). Then if  $\Gamma \vdash_f M : A$  and  $g \geq f$  then  $\Gamma \vdash_g M : A$ .
- *Variable Weakening:* If  $\Gamma \vdash_f M : A$  and  $y$  does not occur in  $\Gamma$ , then  $\Gamma, y : B \vdash_{f+0y} M : A$ .
- *Substitution:* If  $\Gamma \vdash_f M : A$  and  $\Gamma, x : A \vdash_g N : B$ , then  $\Gamma \vdash_{g[f/x]} N[M/x] : B$ .

First, we can weaken the resource subscript, allowing more uses of a variable or more credits in the bank (e.g. if  $\cdot \vdash_3 M : A$ , then  $\cdot \vdash_5 M : A$ ). Second, we can weaken a context to include an unused variable (we write  $f + 0y$  for emphasis, but by equating resource terms up to arithmetic identities, this is just  $f$ ). Third, we can substitute one term into another, performing the corresponding substitution on resource terms. The idea is that, if  $N$  uses a variable  $x$  say 3 times, then it requires 3 times the resources needed to make  $M$  to duplicate  $M$  three times; this multiplication occurs when substituting  $f$  for the occurrence of  $x$  in  $g$ .

**2.1.2 Multiplicative/Additive Rules in  $n$ -linear Style.** In the  $n$ -linear types style of presentation, rules of linear logic that traditionally split the context (e.g.  $\otimes$  introduction,  $\multimap$  elimination) sum the resources used in each premise, but keep the same underlying variable context  $\Gamma$  in all premises.

Admissible:	$\frac{\Gamma \vdash_f M : A \quad g \geq f}{\Gamma \vdash_g M : A}$	$\frac{\Gamma \vdash_f M : A}{\Gamma, y : B \vdash_{f+0y} M : A}$	$\frac{\Gamma \vdash_f M : A \quad \Gamma, x : A \vdash_g N : B}{\Gamma \vdash_{g[f/x]} N[M/x] : B}$
$\frac{}{\Gamma, x : A \vdash_{f+x} x : A}$	$\frac{\Gamma \vdash_f M : A}{\Gamma \vdash_f \text{tick} ; M : A}$	$\frac{\Gamma \vdash_{f+\ell} M : A}{\Gamma \vdash_f \text{create}_\ell M : A}$	$\frac{\Gamma \vdash_f M : A}{\Gamma \vdash_{f+\ell} \text{spend}_\ell M : A}$
$\frac{\Gamma \vdash_f M : A \quad kf + \ell \leq g}{\Gamma \vdash_g \text{save}_\ell^k M : !_\ell^k A}$	$\frac{\Gamma \vdash_f M : !_\ell^k A \quad \Gamma, x : A \vdash_{g+k'(kx+\ell)} N : B}{\Gamma \vdash_{k'f+g} \text{transfer}_{k'} !_\ell^k x = M \text{ to } N : B}$		
$\frac{\Gamma, x : A \vdash_{f+x} M : B}{\Gamma \vdash_f \lambda x. M : A \multimap B}$	$\frac{\Gamma \vdash_f M : A \multimap B \quad \Gamma \vdash_g N : A}{\Gamma \vdash_{f+g} M N : B}$		
$\frac{\Gamma \vdash_f M : A}{\Gamma \vdash_f \text{inl } M : A \oplus B}$	$\frac{\Gamma \vdash_f M : B}{\Gamma \vdash_f \text{inr } M : A \oplus B}$	$\frac{\Gamma \vdash_f M : A \oplus B \quad \Gamma, x : A \vdash_{g+k'x} N_1 : C \quad \Gamma, y : B \vdash_{g+k'y} N_2 : C}{\Gamma \vdash_{k'f+g} \text{case}_{k'} (M, x.N_1, y.N_2) : C}$	
	$\frac{\Gamma \vdash_f M : A \quad \Gamma \vdash_f N : B}{\Gamma \vdash_f \langle M, N \rangle : A \& B}$	$\frac{\Gamma \vdash_f M : A_1 \& A_2}{\Gamma \vdash_f \pi_i M : A_i}$	
$\frac{}{\Gamma \vdash_f () : 1}$	$\frac{\Gamma \vdash_f M : A \quad \Gamma \vdash_g N : B}{\Gamma \vdash_{f+g} (M, N) : A \otimes B}$	$\frac{\Gamma \vdash_f M : A \otimes B \quad \Gamma, x : A, y : B \vdash_{g+k'(x+y)} N : C}{\Gamma \vdash_{k'f+g} \text{split}_{k'} (M, x.y.N) : C}$	
$\frac{}{\Gamma \vdash_f 0 : \mathbb{N}}$	$\frac{\Gamma \vdash_f M : \mathbb{N}}{\Gamma \vdash_f S(M) : \mathbb{N}}$	$\frac{\Gamma \vdash_f M : \mathbb{N} \quad \Gamma \vdash_{g_1} N_1 : 1 \multimap C \quad \Gamma \vdash_{g_2} N_2 : !_0^\infty (\mathbb{N} \otimes (1 \multimap C) \multimap C)}{\Gamma \vdash_{f+g_1+g_2} \text{nrec} (M, N_1, N_2) : C}$	
		$\frac{\Gamma \vdash_f M : \text{List}(A) \quad \Gamma \vdash_{g_1} N_1 : 1 \multimap C \quad \Gamma \vdash_{g_2} N_2 : !_0^\infty (A \otimes (\text{List}(A) \& C) \multimap C)}{\Gamma \vdash_{f+g_1+g_2} \text{lrec} (M, N_1, N_2) : C}$	
$\frac{}{\Gamma \vdash_f [] : \text{List}(A)}$	$\frac{\Gamma \vdash_f M_1 : A \quad \Gamma \vdash_g M_2 : \text{List}(A)}{\Gamma \vdash_{f+g} M_1 :: M_2 : \text{List}(A)}$		

Fig. 2.  $\lambda^A$  Typing Rules

For example, in a positive pair  $(M, N) : A \otimes B$ , if  $M$  is allowed to use  $x$  3 times and  $N$  is allowed to use  $x$  4 times, then the whole pair must be allowed to use  $x$  7 times. As a special case, if a variable is not allowed to occur in, e.g.,  $N$ , it can be marked with a coefficient of 0. On the other hand, rules for additives (e.g. pairing for  $A \& B$ ) use the same resource term in multiple premises. While the elimination rule for  $\oplus$  is additive in sequent calculus style, in natural deduction there is some summing because it builds in a cut for the term being case-analyzed.

**2.1.3 Ticks, and Creating/Spending Credits.** The  $\text{tick} ; M$  construct is used to mark program points that are intended to incur one unit of time cost (e.g. bit flips in the binary counter example); it uses the same resources as  $M$ .

$\text{create}$  is the means to create credits, where  $\text{create}_\ell$  gives  $M$  access to  $\ell$  extra credits to use, along with whatever resources are present in the ambient context; formally, this is represented by adding to the “bank” in the premise of the typing rule for  $M$ . In the operational semantics and



recurrence extraction below, create adds  $\ell$  steps to the amortized cost of  $M$ —it is used to “prepay” for later costs.

$\text{spend}$  is the means to spend credits, where  $\text{spend}_\ell$  spends  $\ell$  credits; because credits can only be spent once, these  $\ell$  credits in the conclusion of the typing rule are not also available in the premise for  $M$ . In the operational semantics/recurrence extraction,  $\text{spend}$  subtracts  $\ell$  steps from the amortized cost of  $M$ —it is used to take advantage of prepaid steps. Note that  $\text{spend}$  satisfies the same typing judgments as an instance of resource weakening (because  $f + \ell \geq f$ ); the “silent” weakening does not change the amortized cost, but instead is a case where our recurrence extraction might obtain a non-tight upper-bound.

**2.1.4  $!_\ell^k$  Modality.** Instead of having two separate modalities, one for  $n$ -use types and the other for types storing credits, we combine them into a single modality  $!_\ell^k A$ . A value of type  $!_\ell^k A$  is a  $k$ -use  $A$  with  $\ell$  credits attached (not  $k \cdot \ell$  credits, which is what one would expect if each use had  $\ell$  credits attached—though that could be modeled by the type  $!_0^k (!_\ell^1 A)$ ). While we write  $a$  and  $\ell$  for nonnegative numbers or  $\infty$ , we restrict  $k$  to range over a *positive* number or  $\infty$ —i.e. we do not allow a “zero-use” modality  $!_\ell^0 A$ , which would complicate the erasure of  $\lambda^A$  to regular simply typed lambda calculus.

The introduction rule for  $!_\ell^k$  says that if we can prove  $M$  has type  $A$  with  $f$  resources, then a version of  $M$  that can be used  $k$  times requires  $kf$  resources. If in addition,  $\ell$  credits are to be attached, then  $kf + \ell$  resources are required. Intuitively, one can think of  $\text{save}_\ell^k M$  as the act of running  $M$  once to obtain its value, but repeating whatever requirement it imposes on the bank  $k$  times, which justifies making  $k$  uses of its value, and then attaching  $\ell$  credits to this value. In order to make resource weakening admissible in general, it is necessary to build weakening into this rule (the second premise).

The elimination rule for the modality allows for the credit stored on a term to be released into the ambient context of another in order to be redistributed or spent. We first present a simplified version, and then explain the general version. Given  $\Gamma \vdash_f M : !_\ell^k A$ , we essentially have  $k$  copies of an  $A$ , along with  $\ell$  extra credits. Given a term  $N$  which can use  $k$  copies of an  $A$  and  $\ell$  credits,  $\Gamma, y : A \vdash_{ky+\ell} N : C$ , we can form the term  $\Gamma \vdash_f \text{transfer } !_\ell^k y = M \text{ to } N : C$ , which, intuitively, deconstructs  $M$  into its  $k$ -usable value and  $\ell$  credits, and moves them to  $N$ , where they can be used. On top of this version, we make two modifications. Firstly,  $N$  should have access to resources other than just what’s provided to it by  $M$ —so we add a resource term  $g$  available in  $N$  (and therefore required to type the transfer). Secondly, it may be necessary at the site of the transfer to further duplicate the  $M : !_\ell^k A$ —this is required to prove a fusion law below, for example. To support this, we parameterize the transfer term by another number,  $k'$ , arriving at the version of the rule presented in Figure 2, which should be thought of as eliminating  $k'$  copies of a  $!_\ell^k A$  at once. The rules for other positive types ( $\oplus, \otimes$ ) similarly permit elimination of multiple copies at once.

The  $!$  modality satisfies the following interactions with other logical connectives, where we write  $A \dashv\vdash B$  to mean interprovability/functions in both directions:

THEOREM 2.2 (FUSION LAWS).

- (1)  $!_{\ell_1+k_1 \cdot \ell_2}^{k_1 k_2} A \dashv\vdash !_{\ell_1}^{k_1} !_{\ell_2}^{k_2} A$
- (2)  $!_{\ell_1+\ell_2}^k (A \otimes B) \dashv\vdash !_{\ell_1}^k A \otimes !_{\ell_2}^k B$
- (3)  $!_\ell^k (A \oplus B) \dashv\vdash !_\ell^k A \oplus !_\ell^k B$

**2.1.5 Natural number recursor.** For natural numbers, while the rules for zero and successor are standard, the recursor takes a bit of explanation. We think of the recursor  $\text{nrec}$  as a function constant of type  $\mathbb{N} \multimap (1 \multimap C) \multimap !_0^\infty (\mathbb{N} \times (1 \multimap C) \multimap C) \multimap C$ . The base case is “thunked”

because we think of  $\multimap$  as a call-by-value function type, but the base case should not be evaluated until the recurrence argument is 0. The ordinary type for the step function (inductive case) would be  $(\mathbb{N} \times C \multimap C)$ , but we also suspend the recursive call, to allow for a simple case analysis that chooses not to use the recursive call. The  $!_0^\infty$  modality surrounding the step function is needed to ensure that the step function itself does not use any ambient credits, which is necessary because the step function is applied repeatedly by the recursor ( $n$  times if the value of  $M$  is  $n$ ). Without this restriction, one could, for example, iterate a step function that spends  $k$  credits to subtract  $Mk$  credits from the amortized cost, while only having  $k$  credits in the bank to spend. For example, without the use of  $!_0^\infty$ , the term  $\cdot \vdash_1 \text{nrec}(7, \lambda\_ . 0, \lambda\_ . \text{spend}_1 0) : \mathbb{N}$  typechecks with only one credit in the ambient bank, but intuitively subtracts 7 from the amortized cost, rather than just the 1 credit that was allowed. We solve this problem using the type  $!_0^\infty A$  (where  $A$  is the ordinary type of the step function  $\mathbb{N} \otimes (1 \multimap C) \multimap C$ ), which represents an infinitely duplicable  $A$  that stores no additional credits. Being infinitely duplicable is an over-approximation, because the step function really only needs to be run  $M$  times, but being more precise would require reasoning about such values in the type system.

In the common case, the step function will use other infinite-use variables but no credits from the bank. A typical typing derivation for this case, where  $H$  is the type of a helper function and  $A$  is the type of the step function, would be

$$\frac{f : H \vdash_{\infty f} N'_2 : A}{f : H \vdash_{\infty(\infty f)=\infty f} \text{save}_0^\infty N'_2 : !_0^\infty A}$$

Using this as the third premise of the typing rule of  $\text{nrec}$ , we see that such an  $\text{nrec}$  itself requires only the credits demanded by the number argument ( $M$ ) and base case ( $N_1$ ), assuming  $f$  is substituted by a helper function that uses no credits.

The way in which the  $!_0^\infty$  modality “prevents” the use of credits from the bank is somewhat subtle: a step function *can* use credits from the bank, but this will require the bank to be infinite in the conclusion. This is because the introduction rule for  $!_0^\infty$  inflates any finite resources to  $\infty$  in the conclusion:

$$\frac{f : H \vdash_{2x+3} N'_2 : A}{f : H \vdash_{\infty(2f+3)=\infty f+\infty} \text{save}_0^\infty N'_2 : !_0^\infty A}$$

Thus, the step function is only permitted to use credits from the bank when the bank has  $\infty$  credits in the conclusion, while we are generally interested in programs that use finitely many credits.

**2.1.6 List recursor.** The list recursor  $\text{lrec}(M, N_1, N_2)$  has the same “credit capture” problem as the recursor on naturals, which we solve using  $!_0^\infty$ . The list recursor has another challenge, though, because unlike a natural number, the values of the list can themselves store credits. Because of this, to prevent credits from being duplicated, in the cons case, the recursor may use *either* the tail of the list or the recursive result, but not both. We code this using an internal choice/negative product  $\&$ . The negative product will itself be treated as a lazy type constructor, where an  $A \& B$  pair is a value even when the  $A$  and  $B$  are not, so we do not need to further thunk the recursive result  $C$  here.

## 2.2 Operational Semantics for $\lambda^A$

We present a call-by-value big-step operational semantics for  $\lambda^A$  in Figure 3, whose primary judgment form is  $M \downarrow^{(n,r)} v$ , which means that  $M$  evaluates to the value  $v$  with cost  $(n, r)$ . The first component of the cost,  $n$  (a non-negative number) indicates the *real cost* of evaluating  $M$ , in this case the number of ticks performed while evaluating  $M$ . The second component,  $r$  (which can be any integer), tracks creates and spends — the (possibly negative) sum total of credits created and spent while evaluating  $M$ , where creating is positive and spending is negative. The *amortized cost*



of evaluating  $M$  is  $n + r$ : the number of “actual” steps taken, plus the number of credits created, minus the number spent.

One reason we separate  $n$  and  $r$  in the judgment form is that there is a straightforward *erasure* of  $\lambda^A$  to ordinary simply typed  $\lambda$ -calculus (STLC with a tick operation), in which evaluating the STLC program has cost (number of ticks)  $n$ . Briefly, this translation translates  $!_\ell^k A$  to  $A$ , translates all of the linear connectives to their unrestricted counterparts, drops all create, spend, save term constructors, and translates transfer to a let. The definition of  $n$  in each of our inference rules for  $M \Downarrow^{(n,r)} v$  is the same as the usual cost for STLC with a tick operation, so this erasure preserves cost. Because of this erasure, the  $n$  in  $M \Downarrow^{(n,r)} v$  is a meaningful cost to bound. Further, the distinction between  $n$  and  $r$  is why we have separate terms create and tick: tick increases the operational cost which should be preserved under erasure, while create increase the amortized cost only.

As discussed in Section 2.1.3,  $\text{create}_\ell M$  creates  $\ell$  credits for  $M$  to use for the price of  $\ell$  units of time cost, whereas  $\text{spend}$  subtracts from the amortized cost of an expression — a speedup which is paid for by the  $\ell$  credits which the body is no longer allowed to use. Both are reflected by corresponding changes to  $r$ .

The operational intuition for  $\text{save}_\ell^k M : !_\ell^k A$  is that it runs  $M$  once, but repeats whatever effect this had on the credit bank  $k$  times, which justifies using the credits in the value of  $M$   $k$  times. (The erasure to STLC discussed above runs  $M$  only once, not  $k$  times—which would be challenging when  $k$  is  $\infty$ .) Formally, this means that the  $n$  in the conclusion is just the  $n$  in the premise, but the  $r$  is multiplied by  $k$ . Running  $\text{save}_\ell^k$  does *not* add  $\ell$  to the  $r$  component because  $\text{save}$  does not create credits (adding to the amortized cost), but only attaches some already existing credits to the value  $v$ . Recall that  $\text{transfer}$  detaches the credits from a  $!_\ell^k$  value, and allows for them, along with the  $k$  copies of the value, to be used in another term. The evaluation rule says that, in order to evaluate  $\text{transfer}_{k'} !_\ell^k x = M$  to  $N$ , we first evaluate  $M$  to a save value, and then evaluate the substitution instance  $N[v_1/x]$ . The  $k'$  in  $\text{transfer}$  means to repeat the evaluation of  $M$   $k'$  times, allowing  $k \cdot k'$  uses in the body of  $N$ , so this (similarly to  $\text{save}$ ) repeats the credit effects  $r_1$  of  $M$   $k'$  times in the conclusion. The other positive elimination forms are similar.

### 2.3 Syntactic Properties

In the operational semantics judgment  $M \Downarrow^{(n,r)} v$ , we think of  $n + r$  (the actual cost  $n$  plus the credit difference  $r$ ) as the amortized cost of the program. A key property of amortized analysis is that the amortized cost is an upper bound on the true cost, which means in this case that  $n + r \geq n$ , so we would like  $r \geq 0$ . While  $r$  is in general allowed to be a negative number, it is controlled by the credits  $a$  of the typing judgment  $\cdot \vdash_a M : A$ , intuitively because it is only  $\text{spend}$  operations that subtract from  $r$ , and  $\text{spend}$  operations are only allowed when the type system deems there to be sufficient credits available. Thus, we will be able to prove that  $r \geq 0$  for well-typed terms. To do so, we strengthen the induction hypotheses to prove that  $\cdot \vdash_a M : A$  and  $M \Downarrow^{(n,r)} v$  imply  $a + r \geq 0$ , which gives  $r \geq 0$  for closed programs that use no external credits (so  $a = 0$ ), which is what a “main” function is expected to be (e.g. set in the binary counter example). It is technically convenient to combine this with a preservation result, stating that the credits of  $v$  is in fact  $a + r$  (the resource term in a typing judgment must be non-negative, so  $a + r \geq 0$  is in fact a prerequisite for even asserting that  $\cdot \vdash_{a+r} v : A$ ).

**THEOREM 2.3 (PRESERVATION BOUND).** *If  $\cdot \vdash_a M : A$  and  $M \Downarrow^{(n,r)} v$ , then  $a + r \geq 0$  and  $\cdot \vdash_{a+r} v : A$ .*

We also have that values evaluate in 0 steps:

**THEOREM 2.4.** *If  $v$  is a value, and  $v \Downarrow^{(n,r)} v$ , then  $n = r = 0$ .*

and that values of type  $\mathbb{N}$  contain no credits:

$$\begin{array}{c}
\frac{M \downarrow^{(n,r)} v}{\text{tick} ; M \downarrow^{(1+n,r)} v} \quad \frac{M \downarrow^{(n,r)} v}{\text{create}_\ell M \downarrow^{(n,r+\ell)} v} \quad \frac{M \downarrow^{(n,r)} v}{\text{spend}_\ell M \downarrow^{(n,r-\ell)} v} \\
\\
\frac{M \downarrow^{(n,r)} v}{\text{save}_\ell^k M \downarrow^{(n,kr)} \text{save}_\ell^k v} \quad \frac{M \downarrow^{(n_1,r_1)} \text{save}_\ell^k v_1 \quad N[v_1/x] \downarrow^{(n_2,r_2)} v}{\text{transfer}_{k'} !_\ell^k x = M \text{ to } N \downarrow^{(n_1+n_2,k'r_1+r_2)} v} \\
\\
\frac{}{\lambda x.M \downarrow^{(0,0)} \lambda x.M} \quad \frac{M \downarrow^{(n_1,r_1)} \lambda x.M' \quad N \downarrow^{(n_2,r_2)} v_1 \quad M'[v_1/x] \downarrow^{(n_3,r_3)} v}{M N \downarrow^{(n_1+n_2+n_3,r_1+r_2+r_3)} v} \\
\\
\frac{M \downarrow^{(n,r)} v}{\text{inr } M \downarrow^{(n,r)} \text{inr } v} \quad \frac{M \downarrow^{(n_1,r_1)} \text{inr } v_1 \quad N_2[v_1/x] \downarrow^{(n_2,r_2)} v}{\text{case}_{k'}(M, x.N_1, y.N_2) \downarrow^{(n_1+n_2,k'r_1+r_2)} v} \\
\\
\frac{M \downarrow^{(n,r)} v}{\text{inl } M \downarrow^{(n,r)} \text{inl } v} \quad \frac{M \downarrow^{(n_1,r_1)} \text{inl } v_1 \quad N_1[v_1/x] \downarrow^{(n_2,r_2)} v}{\text{case}_{k'}(M, x.N_1, y.N_2) \downarrow^{(n_1+n_2,k'r_1+r_2)} v} \\
\\
\frac{}{\langle M, N \rangle \downarrow^{(0,0)} \langle M, N \rangle} \quad \frac{M \downarrow^{(n_1,r_1)} \langle N_1, N_2 \rangle \quad N_i \downarrow^{(n_2,r_2)} v}{\pi_i M \downarrow^{(n_1+n_2,r_1+r_2)} v} \\
\\
\frac{M \downarrow^{(n_1,r_1)} v_1 \quad N \downarrow^{(n_2,r_2)} v_2}{(M, N) \downarrow^{(n_1+n_1,r_1+r_2)} (v_1, v_2)} \quad \frac{M \downarrow^{(n_1,r_1)} (v_1, v_2) \quad N[v_1/x, v_2/y] \downarrow^{(n_2,r_2)} v}{\text{split}_{k'}(M, x.y.N) \downarrow^{(n_1+n_2,k'r_1+r_2)} v} \\
\\
\frac{}{0 \downarrow^{(0,0)} 0} \quad \frac{M \downarrow^{(n,r)} v}{S(M) \downarrow^{(n,r)} S(v)} \quad \frac{}{() \downarrow^{(0,0)} ()} \\
\\
\frac{M \downarrow^{(n_1,r_1)} 0 \quad N_1 \downarrow^{(n_2,r_2)} \lambda x.N'_1 \quad N_2 \downarrow^{(n_3,r_3)} v' \quad N'_1[()/x] \downarrow^{(n_4,r_4)} v}{\text{nrec}(M, N_1, N_2) \downarrow^{(n_1+n_2+n_3+n_4,r_1+r_2+r_3+r_4)} v} \\
\\
\frac{M \downarrow^{(n_1,r_1)} S(v_1) \quad N_2 \downarrow^{(n_2,r_2)} \text{save}_0^\infty (\lambda x.N'_2) \quad N_1 \downarrow^{(n_3,r_3)} \lambda x.N'_1 \quad N'_2[(v_1, \lambda z.(\text{nrec}(v_1, \lambda x.N'_1, \text{save}_0^\infty (\lambda x.N'_2))))/x] \downarrow^{(n_4,r_4)} v}{\text{nrec}(M, N_1, N_2) \downarrow^{(n_1+n_2+n_3+n_4,r_1+r_2+r_3+r_4)} v} \\
\\
\frac{M \downarrow^{(n_1,r_1)} [] \quad N_1 \downarrow^{(n_2,r_2)} \lambda x.N'_1 \quad N_2 \downarrow^{(n_3,r_3)} \text{save}_0^\infty (\lambda x.N'_2) \quad N'_1[()/x] \downarrow^{(n_4,r_4)} v}{\text{lrec}(M, N_1, N_2) \downarrow^{(n_1+n_2+n_3+n_4,r_1+r_2+r_3+r_4)} v} \\
\\
\frac{M \downarrow^{(n_1,r_1)} v_1 :: v_2 \quad N_2 \downarrow^{(n_2,r_2)} \text{save}_0^\infty (\lambda x.N'_2) \quad N_1 \downarrow^{(n_3,r_3)} \lambda x.N'_1 \quad N'_2[(v_1, \langle v_2, \text{lrec}(v_2, \lambda x.N'_1, \text{save}_0^\infty (\lambda x.N'_2)) \rangle)/x] \downarrow^{(n_4,r_4)} v}{\text{lrec}(M, N_1, N_2) \downarrow^{(n_1+n_2+n_3+n_4,r_1+r_2+r_3+r_4)} v}
\end{array}$$

Fig. 3.  $\lambda^A$  Operational Semantics

$$\begin{aligned}
\cdot \vdash_0 \text{inc} &:= \lambda b. \text{lrec}(b, \lambda\_.\text{tick} ; \text{create}_1 (\text{inl} (\text{save}_1^1 ())) :: [], \\
&\quad \text{save}_0^\infty (\lambda(a, tr). \text{case}_1(a, \_.\text{tick} ; \text{create}_1 (\text{inl} (\text{save}_1^1 ())) :: \pi_1 tr, \\
&\quad \quad y. \text{transfer}_1 !_1^1 = y \text{ to} \\
&\quad \quad \text{spend}_1 (\text{tick} ; \text{inl} () :: \pi_2 tr))) : \text{List}(\text{bit}) \multimap \text{List}(\text{bit}) \\
\vdash_0 \text{set} &:= \lambda n. \text{nrec}(n, \lambda\_.[], \text{save}_0^\infty (\lambda p. \text{split}_1(p, \_.\text{inc} (x ()))))) : \mathbb{N} \multimap \text{List}(\text{bit})
\end{aligned}$$
Fig. 4. Binary Counter Terms in  $\lambda^A$ 

**THEOREM 2.5 (RESOURCE STRENGTHENING FOR  $\mathbb{N}$ ).** *If  $\cdot \vdash_a v : \mathbb{N}$ , then  $\cdot \vdash_0 v : \mathbb{N}$*

## 2.4 Binary Counter Annotation

As an example, we translate the binary counter program from Section 1 to  $\lambda^A$ , decorating the program with `create`, `spend`, `save`, and `transfer` in order to emulate the analysis described in Section 1. Since the analysis stores credits on 1 bits, the type of bits is  $\text{bit} = 1\oplus_1^1 1$ ; a value `inl ()` represents a 0 bit, and a value `inr (save11 ())` represents a 1 bit, with a credit attached. A binary number is represented as a list of bits, `List (bit)`. The cost of interest is the number of bit flips, so we insert `ticks` everywhere a bit is flipped from 0 to 1 or vice versa. Next, to handle the credits, we create and subsequently save a credit when we flip a bit from 0 to 1, and transfer then spend when flipping bits from 0 to 1. This annotation is shown in Figure 4 – for simplicity, we use `inc` as a meta-level name for the term implementing the function, so its occurrence in `set` really means a copy of that entire term (to do this at the object level, we could alternatively think of a top-level definition of `inc` as binding an infinite-use variable).

## 3 RECURRENCE LANGUAGE $\lambda^C$ , AMORTIZED RECURRENCE EXTRACTION, AND BOUNDING THEOREM

Next, we define a translation from  $\lambda^A$  into a *recurrence language*  $\lambda^C$ . Unlike  $\lambda^A$ ,  $\lambda^C$  has a fully structural (weakening and contraction) type system, and no special constructs for amortized analysis (it is mostly unchanged from [9, 20]). The recurrence translation takes a function in  $\lambda^A$  to a function that outputs the original function’s cost in  $\lambda^C$ , using a cost type  $\mathbb{C}$  (which we will often take to be integers). Formally,  $\mathbb{C}$  can be any commutative ring with an  $\infty$  element, the typical example being the (“tropical”) max-plus ring on the integers, i.e. integers with addition and binary maxes. Some of the typing rules for  $\lambda^C$  are presented in Figure 5.

Relative to our previous work, the main conceptual change for supporting amortized analysis is that, instead of extracting recurrences for the true cost of a program ( $n$  in  $M \downarrow^{(n,r)} v$ ), we extract recurrences that given an upper bound on the program’s amortized cost  $n + r$ , which is itself a bound on the true cost for programs that use no external credits.

### 3.1 Monadic Translation from $\lambda^A$ to $\lambda^C$

Following [9, 10], a function  $A \multimap B$  in  $\lambda^A$  will be translated to a function  $\langle\langle A \rangle\rangle \rightarrow \mathbb{C} \times \langle\langle B \rangle\rangle$ , where for a  $\lambda^A$  type  $A$ , a value of  $\lambda^C$  type  $\langle\langle A \rangle\rangle$  represents the size of a value in  $\lambda^A$ . Intuitively, this means that a function in  $\lambda^A$  is translated to a function that, in terms of the size of the input, gives the cost of running the function and the size of the output. Generalized to higher-type, “size” is properly viewed as “use-cost;” it is a property that tells us how the value affects the cost of a computation that uses it. In an unfortunate terminological clash, prior work [11] refers to this concept as *potential*, with no intentional connotation of potential functions from the physicist’s method of amortized analysis. In order to keep this work consistent with the sequence of papers it follows, and since  $\lambda^A$  is based on the banker’s method, we will only use “potential” to refer to the use-cost of a value,

$$\begin{array}{c}
\frac{}{\Gamma, x : T \vdash x : T} \quad \frac{k \in \mathbb{Z}}{\Gamma \vdash k : \mathbb{C}} \quad \frac{\Gamma \vdash E_1 : \mathbb{C} \quad \Gamma \vdash E_2 : \mathbb{C}}{\Gamma \vdash E_1 + E_2 : \mathbb{C}} \quad \frac{}{\Gamma \vdash () : 1} \\
\\
\frac{\Gamma \vdash E_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash E_2 : T_1}{\Gamma \vdash E_1 E_2 : T_2} \quad \frac{\Gamma, x : T_1 \vdash E : T_2}{\Gamma \vdash \lambda x. E : T_1 \rightarrow T_2} \quad \frac{\Gamma \vdash E_1 : T_1 \quad \Gamma \vdash E_2 : T_2}{\Gamma \vdash (E_1, E_2) : T_1 \times T_2} \quad \frac{\Gamma \vdash E : T_1 \times T_2}{\Gamma \vdash \pi_i E : T_i} \\
\\
\frac{\Gamma \vdash E : T_1}{\Gamma \vdash \text{inl } E : T_1 + T_2} \quad \frac{\Gamma \vdash E : T_2}{\Gamma \vdash \text{inr } E : T_1 + T_2} \quad \frac{\Gamma \vdash E : T_1 + T_2 \quad \Gamma, x : T_1 \vdash E_1 : T \quad \Gamma, y : T_2 \vdash E_2 : T}{\Gamma \vdash \text{case } (E, x.E_1, y.E_2) : T} \\
\\
\frac{}{\Gamma \vdash 0 : \mathbb{N}} \quad \frac{\Gamma \vdash E : \mathbb{N}}{\Gamma \vdash S(E) : \mathbb{N}} \quad \frac{\Gamma \vdash E : \mathbb{N} \quad \Gamma \vdash E_1 : 1 \rightarrow T \quad \Gamma \vdash E_2 : \mathbb{N} \times T \rightarrow T}{\Gamma \vdash \text{nrec } (E, E_1, E_2) : T} \\
\\
\frac{}{\Gamma \vdash [] : \text{List } (T)} \quad \frac{\Gamma \vdash E_1 : T \quad \Gamma \vdash E_2 : \text{List } (T)}{\Gamma \vdash E_1 :: E_2 : \text{List } (T)} \quad \frac{\begin{array}{c} \Gamma \vdash E : \text{List } (T_1) \\ \Gamma \vdash E_1 : 1 \rightarrow T \\ \Gamma \vdash E_2 : T_1 \times (\text{List } (T_1) \times T) \rightarrow T \end{array}}{\Gamma \vdash \text{lrec } (E, E_1, E_2) : T}
\end{array}$$

Fig. 5. Recurrence Language  $\lambda^{\mathbb{C}}$  Definition

and so call  $\langle\!\langle A \rangle\!\rangle$  the *potential type* for  $A$  and a value of type  $\langle\!\langle A \rangle\!\rangle$  a *potential*. The size of the output is needed for the translation to be compositional: the recurrence extracted for a term should be composed of the recurrences extracted for its subterms, but the cost of e.g. a function application depends on the size of the argument itself, not just its cost. A recurrence extraction of this form can be packaged as a monadic translation into the writer monad  $\mathbb{C} \times A$ . We follow [9] in deferring the abstraction of values as sizes to denotational semantics of  $\lambda^{\mathbb{C}}$  defined in Section 4, which allows the same recurrence extraction and bounding theorem to be reused for multiple models with more or less precise notions of size.

We call the pair of a cost and a potential a *complexity*. The translation consists of three separate functions, the definitions of which are shown in Figure 6. Firstly,  $\langle\!\langle \cdot \rangle\!\rangle$  takes a type  $A$  in  $\lambda^A$  and maps it to the type  $\langle\!\langle A \rangle\!\rangle$  whose elements are the potentials of type  $A$ . We extend this to contexts pointwise:  $\langle\!\langle \Gamma, x : A \rangle\!\rangle = \langle\!\langle \Gamma \rangle\!\rangle, x : \langle\!\langle A \rangle\!\rangle$ . The second is  $\|A\| := \mathbb{C} \times \langle\!\langle A \rangle\!\rangle$ , which takes a type  $A$  to the corresponding type of complexities. Finally, we overload  $\|\cdot\|$  to denote the recurrence extraction function from terms of  $\lambda^A$  to terms in  $\lambda^{\mathbb{C}}$ . For convenience, when  $E : \mathbb{C} \times T$ , we often write  $\pi_1 E$  as  $E_c$  (cost) and  $\pi_2 E$  as  $E_p$  (potential).<sup>2</sup> We also use special notation for adding a cost to a complexity, writing  $E +_c E'$  for  $(E + E'_c, E'_p)$  when  $E : \mathbb{C}$  and  $E' : \mathbb{C} \times T$ .

Overall, the idea is that a term is translated to a function from potentials of its context to complexities of its type:

**THEOREM 3.1 (EXTRACTION PRESERVES TYPES).** *If  $\Gamma \vdash_a M : A$  then  $\langle\!\langle \Gamma \rangle\!\rangle \vdash \|M\| : \|A\|$*

We comment on some of the less obvious aspects of this translation:

- $!_{\ell}^k A$ : The type translation erases the  $!_{\ell}^k$  modality.
- $A \& B$ : Since the negative product in  $\lambda^A$  is lazy, a value of type  $A \& B$  is a pair of un-evaluated terms. Thus, the potential of a term of type  $A \& B$  must include the cost of evaluating each term, since that will factor into the cost of using such a value.
- $\text{tick}$ : Since  $\text{tick} ; M$  evaluates with (true cost and) amortized cost 1 higher than  $M$ 's, the cost component of  $\|\text{tick} ; M\|$  is  $1 + \|M\|_c$ .
- $\text{save}_{\ell}^k$ : The extracted amortized cost of  $\text{save}_{\ell}^k M$  is  $k$  times the extracted cost of  $M$ , with the potential remaining the same. This is in principle a non-exact bound, because we are

<sup>2</sup>We regard the subscript notation as binding tighter than ordinary projection: i.e.  $\pi_1 E_p = \pi_1(E_p)$ .

$$\begin{aligned}
\|A\| &= \mathbb{C} \times \langle A \rangle \\
\langle !_{\ell}^k A \rangle &= \langle A \rangle \\
\langle 1 \rangle &= 1 \\
\langle A \otimes B \rangle &= \langle A \rangle \times \langle B \rangle \\
\langle A \oplus B \rangle &= \langle A \rangle + \langle B \rangle \\
\langle A \multimap B \rangle &= \langle A \rangle \rightarrow \|B\| \\
\langle A \& B \rangle &= \|A\| \times \|B\| \\
\langle \mathbb{N} \rangle &= \mathbb{N} \\
\langle \text{List}(A) \rangle &= \text{List}(\langle A \rangle) \\
\|x\| &= (0, x) \\
\|()\| &= (0, ()) \\
\|\text{tick} ; M\| &= 1 +_c \|M\| \\
\|(M, N)\| &= (\|M\|_c + \|N\|_c, (\|M\|_p, \|N\|_p)) \\
\|\text{split}_{k'}(M, x.y.N)\| &= k' \|M\|_c +_c \|N\| [\pi_1 \|M\|_p / x, \pi_2 \|M\|_p / y] \\
\|\lambda x. M\| &= (0, \lambda x. \|M\|) \\
\|M N\| &= (\|M\|_c + \|N\|_c) +_c \|M\|_p \|N\|_p \\
\|\text{inl } M\| &= (\|M\|_c, \text{inl } \|M\|_p) \\
\|\text{inr } M\| &= (\|M\|_c, \text{inr } \|M\|_p) \\
\|\text{case}_{k'}(M, x.N_1, y.N_2)\| &= k' \|M\|_c + \text{case}(\|M\|_p, x. \|N_1\|, y. \|N_2\|) \\
\|0\| &= (0, 0) \\
\|S(M)\| &= (\|M\|_c, S(\|M\|_p)) \\
\|\square\| &= (0, \square) \\
\|M :: N\| &= (\|M\|_c + \|N\|_c, \|M\|_p :: \|N\|_p) \\
\|\text{transfer}_{k'} !_{\ell}^k x = M \text{ to } N\| &= k' \|M\|_c +_c \|N\| [\|M\|_p / x] \\
\|\text{save}_{\ell}^k M\| &= (k \|M\|_c, \|M\|_p) \\
\|\text{create}_{\ell} M\| &= \ell +_c \|M\| \\
\|\text{spend}_{\ell} M\| &= (-\ell) +_c \|M\| \\
\|\langle M, N \rangle\| &= (0, (\|M\|, \|N\|)) \\
\|\pi_i M\| &= \|M\|_c +_c \pi_i (\|M\|_p) \\
\|\text{nrec}(M, N_1, N_2)\| &= (\|M\|_c + \|N_1\|_c + \|N_2\|_c) +_c \text{nrec}(\|M\|_p, \|N_1\|_p, \lambda x. \|N_2\|_p (\pi_1 x, \lambda z. \pi_2 x)) \\
\|\text{lrec}(M, N_1, N_2)\| &= (\|M\|_c + \|N_1\|_c + \|N_2\|_c) +_c \text{lrec}(\|M\|_p, \|N_1\|_p, \lambda x. \|N_2\|_p (\pi_1 x, ((0, \pi_1 \pi_2 x), \pi_2 \pi_2 x)))
\end{aligned}$$

Fig. 6. Recurrence Extraction

conceptually multiplying the operational amortized cost of  $M \downarrow^{(n,r)} v$ , which is  $n + r$ , by  $k$ , whereas the operational semantics gives the more precise  $n + kr$ . We view this as a consequence of the fact that amortized analyses extract recurrences for the amortized cost  $n + r$ , rather than  $n$  and  $r$  separately. However, this inflation is not a problem for our uses of  $!^{\infty}$  in typing recursors because the branches of the recursor are usually values, which have 0 cost, and  $\infty \times 0 = 0$ . In future work, we might consider a recurrence translation into the  $\mathbb{C} \times \mathbb{C} \times A$  monad, with separate extractions of  $n$  and  $r$ , if more precision is needed. This would allow for  $\lambda^A$  to be used in the place of the (linear fragment) of the source language in previous work [9]. Embedding that language into the  $!^{\infty}$  fragment of  $\lambda^A$  and then extracting recurrences into  $\mathbb{C} \times \mathbb{C} \times A$  would yield the same results as applying the non-amortized recurrence extraction. We emphasize that the loss of precision from not making this change has no bearing on *amortized* algorithm analyses, it would only allow for *non-amortized* analyses to also be performed with  $\lambda^A$ —but such analyses are already handled by prior work [9, 22]

$$\begin{array}{c}
C ::= [] \mid \pi_0 C \mid \pi_1 C \mid C E \mid \text{case}(C, x.E, y.E') \mid \text{nrec}(C, E_1, E_2) \mid \text{lrec}(C, E_1, E_2) \\
\\
\frac{\Gamma, x : T' \vdash C[x] : T \quad \Gamma \vdash E_0 \leq_{T'} E_1}{\Gamma \vdash C[E_0] \leq_T C[E_1]} \quad \frac{}{\Gamma \vdash E \leq_T E} \quad \frac{\Gamma \vdash E_1 \leq_T E_2 \quad \Gamma \vdash E_2 \leq_T E_3}{\Gamma \vdash E_1 \leq_T E_3} \\
\\
\frac{}{\Gamma \vdash E_1[E/x] \leq_T \text{case}(\text{inl } E, x.E_1, y.E_2)} \quad \frac{}{\Gamma \vdash E_2[E/x] \leq_T \text{case}(\text{inr } E, x.E_1, y.E_2)} \\
\\
\frac{}{\Gamma \vdash E[E'/x] \leq_T (\lambda x. E) E'} \quad \frac{}{\Gamma \vdash E_i \leq_{T_i} \pi_i(E_1, E_2)} \\
\\
\frac{}{\Gamma \vdash E_1 () \leq_T \text{nrec}(0, E_1, E_2)} \quad \frac{}{\Gamma \vdash E_2 (E, \text{nrec}(E, E_1, E_2)) \leq_T \text{nrec}(S(E), E_1, E_2)} \\
\\
\frac{}{\Gamma \vdash E_1 () \leq_T \text{lrec}([], E_1, E_2)} \quad \frac{}{\Gamma \vdash E_2 (E, (E', \text{lrec}(E', E_1, E_2))) \leq_T \text{lrec}(E :: E', E_1, E_2)}
\end{array}$$

Fig. 7. Syntactic Ordering on  $\lambda^{\mathbb{C}}$ 

- **transfer**: A similar imprecision arises with respect to the multiplicity  $k'$  here, but otherwise transfer is translated like a `let`.
- **nrec**: As in the operational semantics, because we think of the recursor as a call-by-value function constant, some cost is in principle incurred for evaluating the branches to function values, though the branches are usually values in practice.
- **lrec**: The type of the step function in a list recursor is  $!_0^\infty(A \otimes (\text{List}(A) \& C) \multimap C)$ , and the potential translation of this type is  $\langle\!\langle A \rangle\!\rangle \times ((\mathbb{C} \times \text{List}(\langle\!\langle A \rangle\!\rangle)) \times (\mathbb{C} \times \langle\!\langle C \rangle\!\rangle)) \rightarrow \mathbb{C} \times \langle\!\langle C \rangle\!\rangle$ . However, this does not match the required type of the step function of the list recursor in  $\lambda^{\mathbb{C}}$ , which must be  $T_1 \times (\text{List}(T_1) \times T_2) \rightarrow T_2$ . Taking  $T_1 = \langle\!\langle A \rangle\!\rangle$  and  $T_2 = \mathbb{C} \times \langle\!\langle C \rangle\!\rangle$ , the translation of the step function additionally requires a  $\mathbb{C}$  input representing the cost of the tail of the list. However, lists are eager, so the step function is always applied to a value, so we can supply 0 cost here.

### 3.2 Recurrence Language Inequality Judgment

$\lambda^{\mathbb{C}}$  has a syntactic inequality judgment  $\Gamma \vdash E_1 \leq_T E_2$  (Figure 7), which intuitively means that the recurrence  $E_1$  is bounded above by  $E_2$ . For now, we include only those inequalities that are necessary to prove the bounding theorem; this allows for the most models of the recurrence language, and additional axioms valid in particular models can be added in order to simplify recurrences syntactically. The necessary axioms are congruence in the principal positions of elimination forms, as well as the fact that  $\beta$ -reducts are bounded above by their redexes. We often omit the context and type subscript from  $\Gamma \vdash E_1 \leq_T E_2$ , writing  $E_1 \leq_T E_2$  or  $E_1 \leq E_2$ , though formally it is a relation on well-typed terms in context. This relation is primarily a technical device to provide closure properties for the bounding relation. Because of this, we omit a more lengthy discussion of the relation here, and refer the reader to the prior work [9] which introduces this type of relation.

### 3.3 Bounding Relation and its Closure Properties

The correctness of the recurrence extraction is stated in terms of a logical relation between terms in  $\lambda^A$  and terms in  $\lambda^{\mathbb{C}}$ . The intended meaning is that the  $\lambda^{\mathbb{C}}$  recurrence term is an upper bound on the  $\lambda^A$  term's cost and potential.

*Definition 3.2 (Bounding Relation).* When  $\cdot \vdash_a M : A$  and  $\cdot \vdash E : \|A\|$ , then  $M \sqsubseteq^{A,a} E$  if and only if, when  $M \downarrow^{(n,r)} v$ ,



- $n \leq E_c - r$
- $v \sqsubseteq_{\text{val}}^{A, a+r} E_p$

When  $\cdot \vdash_a v : A$  and  $\cdot \vdash E : \langle\langle A \rangle\rangle$ , we define  $v \sqsubseteq_{\text{val}}^{A, a} E$  by induction on  $A$ .

- $\text{save}_\ell^k v \sqsubseteq_{\text{val}}^{A, c} E$  if there exists  $d \geq 0$  so that  $kd + \ell \leq c$ , and  $v \sqsubseteq_{\text{val}}^{A, d} E$
- $\lambda x. M \sqsubseteq_{\text{val}}^{A \multimap B, c} E$  if whenever  $v \sqsubseteq_{\text{val}}^{A, d} E'$ , we have that  $M[v/x] \sqsubseteq_{\text{val}}^{B, c+d} E E'$
- $(v_1, v_2) \sqsubseteq_{\text{val}}^{A_1 \otimes A_2, a} E$  if there are  $a_1, a_2$  such that  $a_1 + a_2 = a$  and  $v_i \sqsubseteq_{\text{val}}^{A_i, a_i} \pi_i E$  for  $i \in \{1, 2\}$
- $[] \sqsubseteq_{\text{val}}^{\text{List}(A), a} E$  iff  $[] \leq_{\text{List}(\langle\langle A \rangle\rangle)} E$
- $v_1 :: v_2 \sqsubseteq_{\text{val}}^{\text{List}(A), a} E$  iff there are  $E_1, E_2$  with  $E_1 :: E_2 \leq_{\text{List}(\langle\langle A \rangle\rangle)} E$ , and there are  $a_1, a_2$  such that  $a_1 + a_2 = a$  such that  $v_1 \sqsubseteq_{\text{val}}^{A, a_1} E_1$  and  $v_2 \sqsubseteq_{\text{val}}^{\text{List}(A), a_2} E_2$ .
- $0 \sqsubseteq_{\text{val}}^{\mathbb{N}, a} E$  iff  $0 \leq E$
- $S(v) \sqsubseteq_{\text{val}}^{\mathbb{N}, a} E$  iff there is some  $E'$  such that  $S(E') \leq_{\mathbb{N}} E$ , and  $v \sqsubseteq_{\text{val}}^{\mathbb{N}, a} E'$
- $\text{inl } v \sqsubseteq_{\text{val}}^{A \oplus B, a} E$  if there exists  $E'$  such that  $\text{inl } E' \leq_{\langle\langle A \rangle\rangle} E$  and  $v \sqsubseteq_{\text{val}}^{A, a} E'$ .
- $\text{inr } v \sqsubseteq_{\text{val}}^{A \oplus B, a} E$  if there exists  $E'$  such that  $\text{inr } E' \leq_{\langle\langle B \rangle\rangle} E$  and  $v \sqsubseteq_{\text{val}}^{B, a} E'$ .
- $() \sqsubseteq_{\text{val}}^{1, a} E$  if  $() \leq_1 E$ .
- $\langle M, N \rangle \sqsubseteq_{\text{val}}^{A \& B, a} E$  if  $M \sqsubseteq_{\text{val}}^{A, a} \pi_1 E$ , and  $N \sqsubseteq_{\text{val}}^{B, a} \pi_2 E$ .

We extend the value bounding relation to substitutions pointwise:  $\theta \sqsubseteq_{\text{sub}}^{\Gamma, \sigma} \Theta$  if for all  $x : A \in \Gamma$ ,  $\theta(x) \sqsubseteq_{\text{val}}^{A, \sigma(x)} \Theta(x)$ . Finally, we define the bounding relation for open terms: when  $\Gamma \vdash_f M : A$ , we say that  $M \sqsubseteq E$  if for all  $\theta \sqsubseteq_{\text{sub}}^{\Gamma, \sigma} \Theta$ , we have  $M[\theta] \sqsubseteq_{\text{val}}^{A, f[\sigma]} E[\Theta]$ .

The *term/expression bounding relation*  $M \sqsubseteq_{\text{val}}^{A, a} E$  says first that the cost component of  $E$  is an upper bound on the amortized cost of  $M$ , which is  $n + r \leq E_c$  (since we will eventually be interested in bounding the actual cost of evaluating  $M$ , we write this as  $n \leq E_c - r$ ). Additionally, expression bounding says that the potential component of  $E$  is an “upper bound” on the value that  $M$  evaluates to; this is expressed via a mutually-defined type-varying *value bounding relation*  $M \sqsubseteq_{\text{val}}^{A, a} E$ . The value bounding relation is defined first by induction on the type  $A$ , and the cases for natural numbers and lists have a local induction on the number/list value as well.<sup>3</sup> We write the credit bank  $a$  as a parameter of the bounding relations, but it is a presupposition that this number is the same one that was used to type check  $\cdot \vdash_a \{M, v\} : A$  (because the bounding relation is on closed terms, the resource subscript is just a single number  $a$ ).

We extend the bounding relation to open terms by considering all closing substitutions: a term  $\Gamma \vdash_f M : A$  is bounded by  $E$  if for every substitution  $\theta$  which is bounded pointwise by  $\Theta$  with some credit function  $\sigma$ , then the closed term  $M[\theta]$  is bounded by  $E[\Theta]$  with  $f[\sigma]$  credits. In this definition,  $\sigma$  gives a number of credits  $a_i$  for each variable  $x_i$ , because  $\theta$  is a substitution of closed terms for variables  $(\cdot \vdash_{a_1} v_1 : A_1)/x_1, (\cdot \vdash_{a_2} v_2 : A_2)/x_2, \dots$

### 3.4 Bounding Theorem

As usual for a logical relation, we first require some lemmas about the bounding relation, before a main loop proving the fundamental theorem that terms are related to their extractions.

First, we have an analogue of Theorem 2.5:

**THEOREM 3.3 ( $\mathbb{N}$ -STRENGTHENING).** *For all  $\cdot \vdash_a v : \mathbb{N}$ , if  $v \sqsubseteq_{\text{val}}^{\mathbb{N}, a} E$ , then  $v \sqsubseteq_{\text{val}}^{\mathbb{N}, 0} E$ .*

Second, we can weaken a bound by recurrence language inequality:

<sup>3</sup>In general, it is necessary to define the relations for inductive types inductively [9], but the values of  $\mathbb{N}$  and  $\text{List}(A)$  are simple enough that induction on values suffices here.

$$\begin{aligned}
\cdot \vdash \|\text{inc}\| &:= (0, \lambda bs. \text{lrec}(bs, \lambda_. (2, (\text{inr } ()) :: []), \\
&\quad \lambda p. (\lambda x. \text{case}(\pi_1 x, \\
&\quad \quad \_ . (2 + (\pi_1 \pi_2 x)_c, (\text{inr } ()) :: (\pi_1 \pi_2 x)_p) \\
&\quad \quad \_ . ((\pi_2 \pi_2 x)_c, (\text{inl } ()) :: (\pi_2 \pi_2 x)_p)) \\
&\quad ) (\pi_1 p, ((0, \pi_1 \pi_2 p), \pi_2 \pi_2 p))) : \mathbb{C} \times (\text{List } (1 + 1) \rightarrow \mathbb{C} \times \text{List } (1 + 1)) \\
\cdot \vdash \|\text{set}\| &:= (0, \lambda n. \text{nrec}(n, \lambda_. (0, []), \lambda u. (0, \lambda p. (\pi_2 p ())_c +_c \|\text{inc}\|_p (\pi_2 p ())_p) \\
&\quad (\pi_1 u, \lambda_. \pi_2 u))) : \mathbb{C} \times (\mathbb{N} \rightarrow \mathbb{C} \times \text{List } (1 + 1))
\end{aligned}$$

Fig. 8. Binary Counter Recurrences in  $\lambda^{\mathbb{C}}$ 

**THEOREM 3.4 (WEAKENING).**

- (1) If  $M \sqsubseteq^{A,a} E$ , and  $E \leq_{\|A\|} E'$ , then  $M \sqsubseteq^{A,a} E'$
- (2) If  $v \sqsubseteq_{\text{val}}^{A,a} E$ , and  $E \leq_{\langle A \rangle} E'$ , then  $v \sqsubseteq_{\text{val}}^{A,a} E'$

Next, we have an analogue of resource weakening in Theorem 2.1:

**THEOREM 3.5 (CREDIT WEAKENING).** If  $a_1 \leq a_2$ , then:

- (1) If  $M \sqsubseteq^{A,a_1} E$ , then  $M \sqsubseteq^{A,a_2} E$
- (2) If  $v \sqsubseteq_{\text{val}}^{A,a_1} E$ , then  $v \sqsubseteq_{\text{val}}^{A,a_2} E$

Next, we have inductive lemmas that will be used in the recursor cases of the fundamental theorem:

**THEOREM 3.6 ( $\mathbb{N}$ -RECURSOR).** If  $\lambda x. N'_1 \sqsubseteq_{\text{val}}^{1-\circ C, c_3} E_1$ ,  $\lambda x. N'_2 \sqsubseteq_{\text{val}}^{\mathbb{N} \otimes (1-\circ C) - \circ C, d} E_2$  with  $d \geq 0$ , then  $\forall n \geq 0$ , if  $\bar{n} \sqsubseteq_{\text{val}}^{\mathbb{N}, 0} E$ , then  $\text{nrec}(\bar{n}, \lambda x. N'_1, \text{save}_0^\infty(\lambda x. N'_2)) \sqsubseteq^{C, c_3 + \infty \cdot d} \text{nrec}(E, E_1, \lambda p. E_2(\pi_1 p, \lambda z. \pi_2 p))$

**THEOREM 3.7 (List(A)-RECURSOR).** If  $\lambda x. N'_1 \sqsubseteq_{\text{val}}^{1-\circ C, c_1} E_1$  and  $\lambda x. N'_2 \sqsubseteq_{\text{val}}^{A \otimes (\text{List}(A) \& C) - \circ C, c_2} E_2$ , then for all values  $\cdot \vdash_d v : \text{List}(A)$  such that  $v \sqsubseteq_{\text{val}}^{\text{List}(A), d} E$ , we have that  $\text{lrec}(v, \lambda x. N'_1, \text{save}_0^\infty(\lambda x. N'_2)) \sqsubseteq^{C, c_1 + d + \infty \cdot c_2} \text{lrec}(E, E_1, \lambda x. E_2(\pi_1 x, ((0, \pi_1 \pi_2 x), \pi_2 \pi_2 x)))$

Using these, we prove the main result:

**THEOREM 3.8 (BOUNDING THEOREM).** If  $\Gamma \vdash_f M : A$ , then  $M \sqsubseteq^A \|M\|$

Finally, for terms that use no external credits, the true cost is bounded by the extracted recurrence:

**COROLLARY 3.9 (TRUE COST BOUNDING).** If  $\cdot \vdash_0 M : A$  and  $M \downarrow^{(n,r)} v$  then  $n \leq \|M\|_c$ .

**PROOF.** By Theorem 3.8, we have  $n \leq \|M\|_c - r$ , but by preservation (Theorem 2.3), we have that  $0 + r \geq 0$ , so  $n \leq \|M\|_c$ .  $\square$

### 3.5 Binary Counter Recurrences

As an example, the binary counter program in  $\lambda^A$  (Figure 4) is translated by the recurrence extraction translation to the terms in Figure 8. Next, we will use a denotational semantics of the recurrence language to simplify these recurrences to the desired closed form.

## 4 RECURRENCE LANGUAGE SEMANTICS

The final step of our technique is to simplify recurrences to closed forms. This can be done semantically, in a denotational model of the recurrence languages, or syntactically, by adding axioms to the inequality judgment  $\Gamma \vdash E \leq_T E'$  corresponding to properties true in a particular model. Here, we will work in a denotational model of  $\lambda^{\mathbb{C}}$  in preorders, which mostly follows previous work [9, 10, 20].

#### 4.1 Semantic Interpretation

We describe the semantic interpretation of  $\lambda^{\mathbb{C}}$  in preorders here, and highlight the differences from [20], which gives a similar presentation with mechanized proofs.

The semantics of types and terms is given in Figure 9, omitting function and product types, which are interpreted using the standard cartesian product and exponential objects of preorders. For each type  $A$  of  $\lambda^{\mathbb{C}}$ , we associate a partially ordered set  $\llbracket A \rrbracket$  equipped with a top element ( $\infty$ ) and binary maximums ( $\vee$ ) for which the top element is an annihilator. We write  $1$  for the one-element poset, and  $\mathbb{N} \cup \infty$  for the natural numbers with an infinite element added, with the usual  $0 \leq 1 \leq 2 \leq \dots \leq \infty$  total order, and  $\mathbb{Z} \cup \infty$  for the integers with an infinite element added, with the usual total order. We write  $P \times Q$  for the cartesian product of posets with the pointwise order, and  $Q^P$  for the poset of monotone functions from  $P$  to  $Q$ , ordered pointwise; these have binary maxes and top elements given pointwise. We write  $P + Q/\sim$  for the “coalesced” sum, which first takes the disjoint union of  $P$  and  $Q$ , with only  $\text{inl}(x) \leq \text{inl}(y)$  if  $x \leq_P y$  and similarly for  $\text{inr}$ , and then equates  $\text{inl}(\infty_P)$  and  $\text{inr}(\infty_Q)$  to create a top element  $\infty_{P+Q/\sim}$ ; binary maxes are defined using maxes in  $P$  and  $Q$  for two elements whose injections match, and to be  $\infty$  otherwise. The translation on types is extended to contexts:  $\llbracket \cdot \rrbracket = 1$ ,  $\llbracket \Gamma, x : A \rrbracket = \llbracket \Gamma \rrbracket \times \llbracket A \rrbracket$ . Finally, we interpret terms of  $\lambda^{\mathbb{C}}$  as *monotone* (but not necessarily infinity- or max-preserving) maps<sup>4</sup> from the interpretation of their contexts into the interpretation of their types. These maps are morphisms in the category **Poset** of partially ordered sets and monotone maps, and so we write them as elements of  $\text{Hom}_{\mathbf{Poset}}(A, B)$ , the set of monotone maps between posets  $A$  and  $B$ .

In Figure 9, we show some representative cases of the interpretation of terms for sums, natural numbers and lists. For costs, the interpretation of cost constants and addition uses the elements and addition of  $\mathbb{Z} \cup \infty$ . In this model, we interpret both natural numbers and lists as  $\mathbb{N} \cup \infty$ ; for lists, this interprets a list as its length.  $\mathbb{N} \cup \infty$  has a  $0$  element and a monotone successor function  $S$ , where  $S(\infty) = \infty$ ; these are used to interpret  $0$ /the empty list and successor/cons. The elimination forms for positives are more complex, and use some auxiliary monotone functions (which are the morphisms in the category of posets):

**THEOREM 4.1.** *For any posets  $A, B, C, G$  with  $\infty$  and  $\vee$ ,*

- (1)  $\text{snrec} \in \text{Hom}_{\mathbf{Poset}} \left( (C^1)^G \times (C^{\mathbb{N} \times C})^G, C^{G \times \mathbb{N}} \right)$
- (2)  $\text{slrec} \in \text{Hom}_{\mathbf{Poset}} \left( (C^1)^G \times (C^{A \times (\mathbb{N} \times C)})^G, C^{G \times \mathbb{N}} \right)$
- (3)  $\text{scase} \in \text{Hom}_{\mathbf{Poset}} \left( C^{G \times A} \times C^{G \times B}, C^{G \times (A+B)} \right)$

The definition of  $\text{scase}$  is required to respect the quotienting  $\text{inl}(\infty) = \text{inr}(\infty)$ ; by maxing each branch the image of  $\infty$  from the other branch, we obtain  $f(\gamma, \infty) \vee g(\gamma, \infty)$  as the image of both of those. The definition of  $\text{snrec}$  is required to be monotone in the  $0 \leq 1 \leq \dots \leq \infty$  ordering; taking the maximum of the base case and the inductive step achieves this, because it forces the image of  $1$  to dominate the image of  $0$ . The definition of  $\text{slrec}$  is similar; the new question that arises is that, because we have abstracted lists as their lengths, forgetting the elements, we do not have a value for the head of the list to supply to  $g$  (which, when we use this operation, will be the translation of the cons branch given to the  $\lambda^{\mathbb{C}}$  recursor). Here, we always supply  $\infty$  as the head list element, which is sufficient when the analysis really does not require any information about the elements of the list (otherwise, one can make a model where lists are interpreted more precisely than as their lengths [8, 9]).

The interpretation satisfies standard soundness theorems:

<sup>4</sup> We write the composition of maps  $f : A \rightarrow B$  and  $g : B \rightarrow C$  in diagrammatic order,  $f; g : A \rightarrow C$ .

$$\begin{aligned}
\llbracket \mathbb{C} \rrbracket &= \mathbb{Z} \cup \{\infty\} \\
\llbracket \mathbb{N} \rrbracket &= \mathbb{N} \cup \{\infty\} \\
\llbracket \text{List}(T) \rrbracket &= \mathbb{N} \cup \{\infty\} \\
\llbracket T_1 + T_2 \rrbracket &= (\llbracket T_1 \rrbracket + \llbracket T_2 \rrbracket) / \sim \text{ where } \text{inl } \infty \sim \text{inr } \infty \\
\llbracket \Gamma, x : T, \Gamma' \vdash x : T \rrbracket &= \pi_1^k; \pi_2 \text{ where } |\Gamma'| = k \\
\llbracket \Gamma \vdash k : \mathbb{C} \rrbracket &= \text{const}(k) \\
\llbracket \Gamma \vdash E_1 + E_2 : \mathbb{C} \rrbracket &= (\llbracket \Gamma \vdash E_1 : \mathbb{C} \rrbracket, \llbracket \Gamma \vdash E_2 : \mathbb{C} \rrbracket); + \\
\llbracket \Gamma \vdash () : 1 \rrbracket &= \text{const}() \\
\llbracket \Gamma \vdash \text{inl } E : T_1 + T_2 \rrbracket &= \llbracket \Gamma \vdash E : T_1 \rrbracket; \text{inl} \\
\llbracket \Gamma \vdash \text{inr } E : E_1 + E_2 \rrbracket &= \llbracket \Gamma \vdash E : E_2 \rrbracket; \text{inr} \\
\llbracket \Gamma \vdash \text{case}(E, x.E_1, y.E_2) : T \rrbracket &= (1_{\llbracket \Gamma \rrbracket}, \llbracket \Gamma \vdash E : T_1 + T_2 \rrbracket); \text{sccase}(\llbracket \Gamma, x : T_1 \vdash E_1 : T \rrbracket, \llbracket \Gamma, y : T_2 \vdash E_2 : T \rrbracket) \\
\text{sccase} &\in \text{Hom}_{\text{Poset}}(C^{G \times A} \times C^{G \times B}, C^{G \times (A+B)}) \\
\text{sccase}(f, g)(\gamma, \text{inl } a) &= f(\gamma, a) \vee g(\gamma, \infty) \\
\text{sccase}(f, g)(\gamma, \text{inr } b) &= f(\gamma, \infty) \vee g(\gamma, b) \\
\llbracket \Gamma \vdash 0 : \mathbb{N} \rrbracket &= \text{const}(0) \\
\llbracket \Gamma \vdash S(M) : \mathbb{N} \rrbracket &= \llbracket \Gamma \vdash M : \mathbb{N} \rrbracket; S \\
\llbracket \Gamma \vdash \text{nrec}(E, E_1, E_2) : T \rrbracket &= (1_{\llbracket \Gamma \rrbracket}, \llbracket \Gamma \vdash E : \mathbb{N} \rrbracket); \text{snrec}(\llbracket \Gamma \vdash E_1 : 1 \rightarrow T \rrbracket, \llbracket \Gamma \vdash E_2 : \mathbb{N} \times T \rightarrow T \rrbracket) \\
\text{snrec} &\in \text{Hom}_{\text{Poset}}((C^1)^G \times (C^{\mathbb{N} \times C})^G, C^{G \times \mathbb{N}}) \\
\text{snrec}(f, g)(\gamma, 0) &= f(\gamma)() \\
\text{snrec}(f, g)(\gamma, n+1) &= g(\gamma)(n, \text{snrec}(f, g)(\gamma, n)) \vee f(\gamma)() \\
\llbracket \Gamma \vdash [] : \text{List}(A) \rrbracket &= \text{const}(0) \\
\llbracket \Gamma \vdash E_1 :: E_2 : \text{List}(A) \rrbracket &= \llbracket \Gamma \vdash E_2 : \text{List}(A) \rrbracket; S \\
\llbracket \Gamma \vdash \text{lrec}(E, E_1, E_2) : T \rrbracket &= (1_{\llbracket \Gamma \rrbracket}, \llbracket \Gamma \vdash E : \text{List}(T)' \rrbracket); \text{slrec}(\llbracket \Gamma \vdash E_1 : 1 \rightarrow T \rrbracket, \llbracket \Gamma \vdash E_2 : T' \times (\text{List}(T)') \times T \rightarrow T \rrbracket) \\
\text{slrec} &\in \text{Hom}_{\text{Poset}}((C^1)^G \times (C^{A \times (\mathbb{N} \times C)})^G, C^{G \times \mathbb{N}}) \\
\text{slrec}(f, g)(\gamma, 0) &= f(\gamma)() \\
\text{slrec}(f, g)(\gamma, n+1) &= g(\gamma)(\infty, (n, \text{slrec}(f, g)(\gamma, n))) \vee f(\gamma)()
\end{aligned}$$

Fig. 9. Semantic Interpretation Definition

**THEOREM 4.2 (COMPOSITIONALITY).** *If  $\Gamma, x : T_1 \vdash E : T_2$ , and  $\Gamma \vdash E' : T_1$ , then  $\llbracket \Gamma \vdash E[E'/x] : T_2 \rrbracket = (1_{\llbracket \Gamma \rrbracket}, \llbracket \Gamma \vdash E' : T_1 \rrbracket); \llbracket \Gamma, x : T_1 \vdash E : T_2 \rrbracket$*

**THEOREM 4.3 (SOUNDNESS (TERMS)).** *If  $\Gamma \vdash E : T$ , then  $\llbracket \Gamma \vdash E : T \rrbracket \in \text{Hom}(\llbracket \Gamma \rrbracket, \llbracket T \rrbracket)$*

**THEOREM 4.4 (SOUNDNESS (INEQUALITY)).** *If  $\Gamma \vdash E \leq E'$ , then for all  $\gamma \in \llbracket \Gamma \rrbracket$ ,  $\llbracket \Gamma \vdash E : T \rrbracket(\gamma) \leq \llbracket \Gamma \vdash E' : T \rrbracket(\gamma)$*

## 4.2 Binary Counter Conclusion

We interpret the binary counter recurrences from Figure 8 in preorders by unfolding the definitions in Figure 9; the result is shown in Figure 10. For the function `inc`, this yields a monotone map  $\llbracket \llbracket \text{inc} \rrbracket_p \rrbracket \in \text{Hom}(1, \mathbb{N} \rightarrow \mathbb{Z} \times \mathbb{N})$ , which is (essentially) a function from an input list size to the cost of evaluation and the length of the output. For the function `set`, this yields a monotone map

Fig. 10. Binary Counter Recurrences Interpreted

We have boxed the parts of the term that are related to computing the cost. The boxed portions of `inc` express that its amortized cost is 2 on the empty list (to create a 1 bit with a credit), is 2 when the bit is 0, and is exactly the same number of steps as the recursive call when the bit is 1. The boxed portions of `set` express that for zero it costs 0, and for successor it costs the recursive call plus the cost of `inc` on the potential of the output of the recursive call. However, because we will show that `inc` turns out to be constant amortized time, we do not need to bound the potential of the output of `set`. Intuitively, to see that `inc` has constant amortized time, observe that the `slrec` will always supply the  $\infty$  bit as the head of the list, which by definition of the coalesced sum is both true and false, so the case is effectively the maximum of 2 and  $\pi_1\pi_2\pi_1p$ . Thus, we effectively have recurrence where  $T_{\text{inc}}(0) = 2$  and  $T_{\text{inc}}(n) = 2 \vee T_{\text{inc}}(n-1)$ , which solves to  $T(n) = 2$  by induction. Substituting this into the recurrence for `set`, we have essentially  $T_{\text{set}}(0) = 0$  and  $T_{\text{set}}(n) = T_{\text{set}}(n-1) + 2$ , which is of course  $O(n)$ . More formally, we can show by induction that for all  $n \geq 0$ ,  $(\llbracket \text{inc} \rrbracket_p)(n)_c \leq 2$ , and that for all  $n$ ,  $(\llbracket \text{set} \rrbracket_p)(n)_c \leq 2n$ , establishing bounds on these recurrences in this denotational semantics in preorders.

By the bounding theorem (Corollary 3.9), we have that, for the true operational cost  $m$  of evaluating  $\text{set}(n) \downarrow^{(m,r)} v$ , we have  $m \leq_{\mathbb{C}} \|\text{set}\|_p(n)_c$  in terms of the syntactic preorder judgment in  $\lambda^{\mathbb{C}}$ . By the soundness of the interpretation in preorders (Theorem 4.4), we have that  $m \leq_{\mathbb{Z} \sqcup \infty} \llbracket \|\text{set}\|_p \rrbracket(n)_c$  in the preorder model. Therefore, by transitivity, we have  $m \leq 2n$  in the preorder model, so our technique proves that the true operational cost  $m$  of setting the binary counter to  $n$  is in fact  $O(n)$ , as desired.

The version of  $\lambda^A$  described thus far supports amortized analyses where the amount of credit stored on each element of a data structure is fixed (e.g.  $\text{List}(!_2A)$  is a list with 2 credits on each element). However, in some important amortized analyses, different amounts of credit must be stored in different parts of a data structure—e.g. for balanced binary search trees implemented via splay trees [35], the number of credits stored on each node is a function of the size of the subtree rooted at that node. In this section, we show that adding existential quantification over credit amounts to  $\lambda^A$  suffices to analyze such examples, using a portion of splay trees as an example. Using existentials, a value of type  $\exists\alpha. !_\alpha A$  is a value of type  $A$  which carries  $\alpha$  credits, for some  $\alpha$ ; for example, a tree whose elements are of type  $\exists\alpha. !_\alpha \mathbb{N}$  stores a variable number of credits with the number on each node. In keeping with our methodology of doing as much of an analysis as possible in the recurrence language and its semantics, the fact that a particular piece of code uses

$$\begin{array}{c}
\frac{\Delta|\Gamma \vdash_f M : A[c/\alpha] \quad \Delta, \alpha \vdash A \text{ type} \quad \Delta \vdash c \text{ credit}}{\Delta|\Gamma \vdash_f \text{pack}_{\alpha=c} M : \exists \alpha. A} \quad \frac{\Delta|\Gamma \vdash_f M : \exists \alpha. A \quad \Delta, \alpha|\Gamma, x : A \vdash_{g+x} N : C \quad \Delta \vdash C \text{ type}}{\Delta|\Gamma \vdash_{f+g} \text{unpack } (\alpha, x) = M \text{ in } N : C} \\
\\
\frac{M \downarrow^{(n,r)} v}{\text{pack}_{\alpha=\ell} M \downarrow^{(n,r)} \text{pack}_{\alpha=\ell} v} \quad \frac{M \downarrow^{(n_1, r_1)} \text{pack}_{\alpha=\ell} v_1 \quad N[\ell/\alpha, v_1/x] \downarrow^{(n_2, r_2)} v}{\text{unpack } (\alpha, x) = M \text{ in } N \downarrow^{(n_1+n_2, r_1+r_2)} v}
\end{array}$$

Fig. 11. Extension of  $\lambda^A$  with existential types

existentials to implement a desired credit policy will not be tracked by the type system, but proved after recurrence extraction. An alternative approach would be to enrich  $\lambda^A$  with some form of indexed or dependent types to track the sizes of data structures in the type system, but such an extension is not necessary for our approach.

### 5.1 Existential Types in $\lambda^A$

To support existential quantifiers over credits, we extend the main typing judgment to be one of the form  $\Delta|\Gamma \vdash_f M : A$ , where  $\Delta = \alpha_1, \dots, \alpha_n$  is a list of “credit variables”. Any of the  $\alpha_i$  can occur free in the types in  $\Gamma$ , the resource term  $f$ , the term  $M$ , or the type  $A$ . Credit variables  $\alpha$  range over *credit terms*  $c$ , which are (finite) sums of credit variables like  $\alpha, \beta$  and credit constants  $\ell$  — i.e.  $\alpha_1 + \alpha_2 + \dots + \alpha_n + \ell$ . We write  $\Delta \vdash c \text{ credit}$  to mean that a credit term is well-formed from the variables in  $\Delta$ . We consider credit terms up to the usual equations for addition on natural numbers. These credit terms can then be used as the “bank” in resource terms: the resource term  $3x + 2y + (\alpha + 2)$  describes a context where one can use  $x$  3 times,  $y$  twice, and has access to the credit term  $\alpha + 2$  credits. Most importantly, credit terms are now allowed to appear in the subscript of the  $!$  modality (generalizing the natural number constants  $\ell$  allowed above): a term  $\alpha \mid \Gamma \vdash_f M : !_{\alpha} A$  with is an  $A$  with  $\alpha$  credits attached. We add a new type  $\exists \alpha. A$  for existentially quantifying over credit variables. A value of type  $\exists \alpha. A$  is a value of type  $A[c/\alpha]$ , for some credit term  $c$ . Such a value does not store the ability to *use* the credits  $c$  — it stores a number of credits itself. However, combining the existential with the  $!$  modality, a value of type  $\exists \alpha. !_{\alpha} A$  is an  $A$  with  $c$  credits attached, for some credit term  $c$ . The operational semantics is defined for terms with no free credit variables, so its structure remains unchanged.

The typing rules and operational semantics for existential types are presented in Figure 11. The terms for existentials are standard pack/unpack terms. The operational semantics of pack and unpack are also standard; because we only evaluate closed terms, the credit term being packed/unpacked with the value will always be a (closed) natural number  $\ell$ .

The rest of the rules for  $\lambda^A$  are mostly unchanged, so we do not repeat them: they are obtained from the rules in Figure 2 by carrying the credit variable context  $\Delta$  through all of the rules, and, in the  $!_c^k$  modality and the save, transfer, create, and spend terms, the natural number constants  $\ell$  are generalized to credit terms  $c$  constructed from these variables. Finally, since the resource terms may contain free credit variables, the ordering judgment on resource terms must be augmented with a credit variable context, and the ordering itself extended to contain the coefficient-wise ordering on credit variables. The operational semantics for these constructs is unchanged, because closed credit terms are precisely the credit values  $\ell$  used above.

For this extension, substitution and type preservation are stated as follows:

**THEOREM 5.1 (SUBSTITUTION).**

- If  $\Delta \vdash c \text{ credit}$  and  $\Delta, \alpha \vdash c' \text{ credit}$ , then  $\Delta \vdash c'[c/\alpha] \text{ credit}$
- If  $\Delta \vdash c \text{ credit}$  and  $\Delta, \alpha|\Gamma \vdash_f M : A$ , then  $\Delta|\Gamma[c/\alpha] \vdash_{f[c/\alpha]} M[c/\alpha] : A[c/\alpha]$



$$\begin{aligned}
\langle\langle \exists \alpha. A \rangle\rangle &= \$ \times \langle\langle A \rangle\rangle \\
\|\text{pack}_{\alpha=c} M\| &= (\|M\|_c, (c, \|M\|_p)) \\
\|\text{unpack } (\alpha, x) = M \text{ in } N\| &= \|M\|_c + c \ \|N\| [\pi_1 \|M\|_p / \alpha, \pi_2 \|M\|_p / x] \\
\|\text{create}_c M\| &= (\text{to}\mathbb{C}(c) + \|M\|_c, \|M\|_p) \\
\|\text{spend}_c M\| &= (-\text{to}\mathbb{C}(c) + \|M\|_c, \|M\|_p)
\end{aligned}$$

Fig. 12. Recurrence extraction for credit existentials

**THEOREM 5.2 (PRESERVATION).** *If  $\cdot \vdash_a M : A$  and  $M \Downarrow^{(n,r)} v$ , then  $a + r \geq 0$  and  $\cdot \vdash_{a+r} v : A$ .*

## 5.2 Extracting Recurrences for Existentials

Recall that the recurrence extraction in Figure 6 erases the  $!^k_\ell A$  modalities and translates  $\text{create}_\ell M$  and  $\text{spend}_\ell M$  by adding/subtracting  $\ell$  to/from the amortized cost. Since we now allow credit variables  $\alpha$ , such as those coming from unpacking an existential type, in the credit position of  $\text{create}/\text{spend}$ , the recurrence extraction will need to refer to the values chosen for  $\alpha$  in order to know how much to add/subtract to/from the amortized cost. Thus, we add a type  $\$$  to the recurrence language, the values of which are numbers of credits, represented by natural numbers. The credit context  $\Delta$  is translated to recurrence language variables of type  $\$$  (i.e.  $\langle\langle \Delta, \alpha \rangle\rangle = \langle\langle \Delta \rangle\rangle, \alpha : \$$ ), while existential types  $\exists \alpha. A$  are translated to pairs  $\$ \times \langle\langle A \rangle\rangle$ . A simple pair suffices because the  $!$  modality is erased by  $\langle\langle \cdot \rangle\rangle$ , and this is the only place where credit terms can occur in the syntax of types, so all occurrences of  $\alpha$  under the binder are removed, and  $\langle\langle A \rangle\rangle$  is a closed type.

We show the new and changed cases of recurrence extraction in Figure 12. The introduction and elimination rules for  $\exists \alpha. A$  translate to the corresponding introduction and elimination forms for  $\$ \times \langle\langle A \rangle\rangle$ . For  $\text{create}$  and  $\text{spend}$ , in principle, we would like the cost component of  $\text{create}_c M$  to be  $c + \|M\|_c$ , but this will not type check, given that  $c : \$$  but  $\|M\|_c : \mathbb{C}$ . Recalling that costs  $\mathbb{C}$ , though axiomatized as a monoid with some operations, are morally integers, we add a coercion  $\text{to}\mathbb{C} : \$ \rightarrow \mathbb{C}$ , which is morally the inclusion of natural numbers into integers.

**THEOREM 5.3 (EXTRACTION PRESERVES TYPES).** *If  $\Delta | \Gamma \vdash_f M : A$ , then  $\langle\langle \Delta \rangle\rangle, \langle\langle \Gamma \rangle\rangle \vdash \|M\| : \|A\|$*

## 5.3 Bounding Relation and Bounding Theorem

The definition of the bounding relation for values (Definition 3.2) is extended with

- $\text{pack}_{\alpha=\ell} v \sqsubseteq_{\text{val}}^{\exists \alpha. A, a} E$  iff  $\ell \leq_\$ \pi_1 E$  and  $v \sqsubseteq_{\text{val}}^{A[\ell/\alpha], a} \pi_2 E$

Recalling that  $E : \langle\langle \exists \alpha. A \rangle\rangle = \$ \times \langle\langle A \rangle\rangle$ , this simply states that the amount of credit packed by  $\alpha$  is bounded by the amount described by  $\pi_1 E$ , and that the value packed with the credit amount is in fact bounded by  $\pi_2 E$ . We remark that this definition may give the careful reader pause—inducting on a substitution instance of an existential type where the existential variable ranges over *types* leads to well-definedness issues. But, our existential variables range over *credits*, so we may simply regard a closed substitution instance of a type  $\alpha \vdash A$  type as a smaller type than  $A$ .

The definition of the bounding relation for open terms must also be modified to quantify over closing substitutions for the credit context, as well as the term context. First, if  $\omega$  is a substitution of credit amounts  $\ell$  for credit variables, and  $\Omega$  is a substitution of closed terms of type  $\$$  for recurrence language variables, then  $\omega \sqsubseteq^\Delta \Omega$  means that for all  $\alpha \in \Delta$ ,  $\omega(\alpha) \leq_\$ \Omega(\alpha)$ . Then for  $\Delta | \Gamma \vdash_f M : A$  we write  $M \sqsubseteq^A E$  if for all  $\omega \sqsubseteq^\Delta \Omega$  and for all  $\theta \sqsubseteq^{\Gamma[\omega], \sigma} \Theta$ , we have that  $M[\omega, \theta] \sqsubseteq^{A[\omega], f[\omega, \sigma]} E[\Omega, \Theta]$ . Using this notation, the bounding theorem is

**THEOREM 5.4 (BOUNDING THEOREM).** *If  $\Delta | \Gamma \vdash_f M : A$ , then  $M \sqsubseteq^A \|M\|$*

$$\begin{aligned}
N_1 &= \lambda_{-}.\text{pack}_{\alpha=0}(\text{save}_0^1 ()) \\
N_2 &= \lambda(\_, (\alpha, \text{save}_1^\alpha ())).\text{create}_1 (\text{pack}_{\beta=\alpha+1} \text{save}_1^{\alpha+1} ()) \\
\text{spawn}(n) &= \text{nrec} (n, N_1, \text{save}_0^\infty N_2) : \exists \alpha. !_\alpha^1 1
\end{aligned}$$

Fig. 13.  $\lambda^A$  term for the spawn function

and the cases which differ from the original Theorem 3.8 are proved in the supplementary materials.

#### 5.4 Splay Tree Analysis

We now show how to use the above machinery to analyze splay trees. We follow Okasaki's presentation [29], in which the key operation is a  $\text{split} : (A \times \text{tree}(A)) \rightarrow \text{tree}(A) \times \text{tree}(A)$  function that splits a given tree into elements larger and smaller than a given pivot. Insertion, deletion, union, intersection, difference etc. can be all implemented from  $\text{split}$  and a  $\text{join}$  operation that combines two sorted trees where all the elements of the first are less than the elements of the second. Showing that  $\text{split}$  is amortized  $O(\log n)$  time, where  $n$  is the size of the tree, is the most difficult part of the amortized analysis, and implies the desired time bounds for the other operations. The key idea of splay trees is that each access rearranges the tree so that accessing the same element twice in a row is quicker the second time. In Okasaki's presentation, this rearrangement takes place in  $\text{split}$ , which performs a series of tree rotations. These rotations ensure that the amortized cost of  $\text{split}$  (amortized over any sequence of binary search tree operations) is  $O(\log n)$ , even though the tree is not always balanced. The most challenging cases of the code unpack the tree to depth two, and rotate the output if they traverses the same direction twice while searching for the pivot:

$$\begin{aligned}
&\text{split } p (N(x, N(y, a_{11}, a_{12}), N(z, a_{21}, a_{22}))) \mid x \geq p \ \&\& \ y \geq p = \\
&\quad (\text{small}, N(y, \text{big}, N(x, a_{12}, N(z, a_{21}, a_{22})))) \text{ where } (\text{small}, \text{big}) = \text{split } p \ a_{11}
\end{aligned}$$

Okasaki's analysis of  $\text{split}$  maintains the invariant that there are  $\varphi(t) = \lceil \lg(|t| + 1) \rceil$  credits associated with the root of every subtree  $t$  in a splay tree, and uses the potential/physicists method to analyze the amortized cost.

The addition of existentials to  $\lambda^A$  allows us to encode this analysis, by giving  $\text{split}$  the type  $A \otimes \text{tree}(\exists \alpha. !_\alpha^1 A) \multimap \text{tree}(\exists \alpha. !_\alpha^1 A) \otimes \text{tree}(\exists \alpha. !_\alpha^1 A)$ , and using code to maintain the invariant that each of these  $\alpha$ 's are precisely  $\varphi(t)$ .

**5.4.1 Creating variable amounts of credit.** To maintain this invariant, we will sometimes need to create amounts of credit determined by a run-time natural number, like  $\varphi(t)$  for some tree  $t$ —but the primitive  $\text{create}_c \ M$  term allows for waiting only for a credit term  $c$ , which cannot depend on run-time values. However, we can write a recursive loop that spawns a number of credits dependent on a run-time value, and package this as a function  $\text{spawn} : \mathbb{N} \multimap \exists \alpha. !_\alpha^1 1$  such that the  $\alpha$  packed in the result of  $\text{spawn}(n)$  is (the credit term representing)  $n$ . The implementation of  $\text{spawn}$  is shown in Figure 13—at a high level, the term loops  $\text{create}_1$  in a  $\mathbb{N}$ -recursor, using a credit existential as a counter variable. In this example, and throughout this section, we use pattern-matching notation as syntactic sugar for the elimination rules for positive types like  $\exists, !, \otimes$ , with the convention that matching on the result of a thunked recursive call implicitly forces it.

In Section 2.2, we argued that the  $n$  component in the operational cost semantics  $M \Downarrow^{n,r} v$  captures the actual operational cost of an erasure to simply-typed  $\lambda$ -calculus, as long as ticks in  $\lambda^A$  are inserted for each STLC  $\beta$ -redex. Because we do not include any tick terms in  $\text{spawn}$ , its abstract operational cost  $n$  is zero. Thus, to realize this cost semantics,  $\text{spawn}$  must be erased before actually running the program. Fortunately, a simple program optimization suffices to do this: translate  $\lambda^A$  to simply-typed  $\lambda$ -calculus by dropping both the  $\exists$  and  $!$  types and the associated

terms, at which point `spawn` has type  $\mathbb{N} \rightarrow 1$ ; then replace all terms of type 1 with the trivial value. That is, we think of `spawn` as a *ghost loop* — code that is meant for the extracted recurrence, but not intended to actually be run.

**5.4.2 Definition of trees in  $\lambda^A$ .** Extending  $\lambda^A$  with the requisite tree type constructor and its rules follows both previous work [9] and the pattern illustrated with lists above. The type of trees is essentially  $\text{tree}(A) = \text{Emp} \mid \mathbb{N} \text{ of } A \otimes \mathbb{N} \otimes \text{tree}(A) \otimes \text{tree}(A)$ . The  $\mathbb{N}$  argument caches the size of the tree, making the function  $\text{size} : \text{tree}(A) \multimap \mathbb{N} \otimes \text{tree}(A)$  — which projects out that field and then rebuilds the tree<sup>5</sup> — constant time. To support coding the `split` function described above, we directly add a recursor that performs a two-level pattern match, with cases for the empty tree, for a node with one child or the other empty and the other is another node, and for a node with two nodes as children; in the latter case, the recursor provides recursive calls on all four subtrees. The details are in the supplementary materials— see Figure 15.

**5.4.3 Splay Tree Implementation.** We define a *splay tree* to be a binary search tree  $t : \text{tree}(\exists \alpha. !_\alpha^\infty A)$  satisfying the property that if  $\text{size}(t) = n$ , then if  $t = N(\_, m, t_0, t_1)$ , then  $t_0$  and  $t_1$  are splay trees, and for  $\llbracket \|t\|_p \rrbracket = N((\alpha, \_), \_, \_)$ , we have  $\alpha = \phi(n)$ . In other words, the credit invariant holds at each node in the tree. We note that each element of the tree not only carries  $\alpha$  credits, but is also infinitely usable since we are required to compare nodes in the tree more than constantly many times. This causes no issues for the extracted recurrences, because keys in the tree are always values. We then prove a lemma which states that `split` preserves the splay tree property — i.e. that the existentially quantified credits stored in the tree satisfy the desired invariant.

**LEMMA 5.5.** *If  $t : \text{tree}(\exists \alpha. !_\alpha^\infty A)$  is a splay tree and  $\text{split}(t) \downarrow (t_0, t_1)$ , then  $t_0$  and  $t_1$  are also splay trees.*

To illustrate the  $\lambda^A$  term for `split`, we show one key case of the recursor, which corresponds to the snippet given at the beginning of this section and to [29, Theorem 5.2]. For this case, we are in the situation where the root, labeled by  $x$ , has two subtrees,  $y$  with subtrees  $a_{11}, a_{12}$ , and  $z$  with subtrees  $a_{21}, a_{22}$ . If the pivot is less than both  $x$  and  $y$ , we recur on the leftmost subtree  $a_{11}$ , which produces the elements of  $a_{11}$  that are smaller and bigger than the pivot. Then *smaller* contains all the elements of the original tree smaller than the pivot. The elements bigger than the pivot are *bigger* and everything else from the original tree; we combine these together into a new tree, performing a rotation to put  $y$  at the root.

The  $\lambda^A$  version of this term, presented in Figure 14, annotates the above code with some additional information about the sizes of trees, and with some code for manipulating credits. The variables  $x, y, z$  are the values of type  $A$  at the root and its immediate children; these come with existentially-quantified numbers of credits  $\alpha, \beta, \gamma$  ( $\alpha$  credits are stored with  $x$ ,  $\beta$  with  $y$ , and  $\gamma$  with  $z$ ), and also with natural numbers caching the sizes of the subtrees that they are the roots of ( $n_1, n_2, n_3$  respectively). The variables  $a_{ij}$  stand for the four subtrees with their (suspended) recursive call outputs; we write `split( $p, a_{11}$ )` for projecting and forcing the recursive call, and write  $a_{ij}$  for projecting the other subtrees. The credit manipulation involves spending the credits  $\alpha$  and  $\beta$  stored with  $x$  and  $y$  in the input tree (we do not spend  $\gamma$ , because the  $z$  node is left unchanged in the output), calculating the sizes of the new nodes  $t'$  and  $s'$  that will be part of the output, and spawning credits corresponding to  $\phi$  of these sizes. The term presented in Figure 14 is one branch of one of the step functions passed to the `treerec` which forms the outermost structure of `split`.

To analyze splay trees, we pass this  $\lambda^A$  term through recurrence extraction and the preorder semantics and then prove the following:

<sup>5</sup>The tree can be rebuilt because values of type  $\mathbb{N}$  are duplicable— there is a diagonal map  $\mathbb{N} \multimap \mathbb{N} \otimes \mathbb{N}$ . Also, we will often use `size` as a function  $\text{tree}(A) \multimap \mathbb{N}$ , and silently contract the second projection for re-use of the argument.

```

 $\lambda((\alpha, \text{save}_{\alpha}^{\infty} x), n_1, (\beta, \text{save}_{\beta}^{\infty} y), n_2, (\gamma, \text{save}_{\gamma}^{\infty} z), n_3, a_{11}, a_{12}, a_{21}, a_{22}).$ 
  if  $x \geq p \ \&\& \ y \geq p$  then tick; let
     $(small, big) = \text{spend}_{\alpha+\beta}(\text{split}(p, a_{11}))$ 
     $d = N(\text{pack}_{\alpha=\gamma}(\text{save}_{\gamma}^{\infty} z), n_3, a_{21}, a_{22})$ 
     $n_{12} = \text{size}(a_{12})$ 
     $n_{big} = \text{size}(big)$ 
     $t'_{size} = 1 + n_{12} + n_3$ 
     $s'_{size} = 2 + n_{big} + n_{12} + n_3$ 
     $((\alpha', _), (\beta', _)) = (\text{spawn}(\varphi(t'_{size})), \text{spawn}(\varphi(s'_{size})))$ 
     $t' = N(\text{pack}_{\alpha=\alpha'}(\text{save}_{\alpha'}^{\infty} x), t'_{size}, a_{12}, d)$ 
     $s' = N(\text{pack}_{\alpha=\beta'}(\text{save}_{\beta'}^{\infty} y), s'_{size}, big, t')$ 
  in  $(small, s')$  end
else ...

```

Fig. 14. Part of the  $\lambda^A$  term for split

**THEOREM 5.6.** *If  $t : \text{tree}(\exists \alpha. !_{\alpha}^{\infty} A)$  is a splay tree with  $\text{size}(t) = n$ , then for any  $v : A$ ,  $\llbracket \llbracket \text{split}(t, v) \rrbracket_c \rrbracket \leq 1 + 2\varphi(\llbracket \llbracket \text{size} \rrbracket_p(t) \rrbracket) \in O(\lg n)$ .*

**PROOF.** As an example, we show the case for the code in Figure 14. The cost component of the extracted recurrence is

$$1 - \alpha - \beta + \llbracket \llbracket \text{split}(p, a_{11}) \rrbracket \rrbracket + \varphi(1 + n_{12} + n_3) + \varphi(2 + n_{big} + n_{12} + n_3)$$

The 1 comes from the tick;  $\alpha$  and  $\beta$  are subtracted because they are spent; and the  $\varphi$  of the sizes of  $t'$  and  $s'$  are added because they are created. By definition,  $1 + n_{12} + n_3 = \llbracket \llbracket \text{size} \rrbracket_p(t') \rrbracket$  and  $2 + n_{big} + n_{12} + n_3 = \llbracket \llbracket \text{size} \rrbracket_p(s') \rrbracket$ . By the credit invariant,  $\alpha = \varphi(\llbracket \llbracket \text{size} \rrbracket_p(t) \rrbracket)$ , and  $\beta = \varphi(\llbracket \llbracket \text{size} \rrbracket_p(s) \rrbracket)$ , where  $s$  is the subtree of  $t$  rooted at  $y$ . Rewriting by these and commuting terms, the extracted recurrence is precisely

$$1 + \llbracket \llbracket \text{split}(p, a_{11}) \rrbracket \rrbracket + \varphi(\llbracket \llbracket \text{size} \rrbracket_p(s') \rrbracket) + \varphi(\llbracket \llbracket \text{size} \rrbracket_p(t') \rrbracket) - \varphi(\llbracket \llbracket \text{size} \rrbracket_p(s) \rrbracket) - \varphi(\llbracket \llbracket \text{size} \rrbracket_p(t) \rrbracket)$$

which Okasaki [29, Theorem 5.2] proves is bounded by  $1 + 2\varphi(\text{size}(t))$ , as required.  $\square$

## 6 RELATED WORK

Techniques for extracting cost information from high-level program source code is a project that is almost as old as studying programming languages (and here we will not even broach the subject of worst-case execution time (WCET) analysis, which focuses on low-level timing analysis on specific hardware). For functional languages, we have examples from the 1970s and 1980s by Wegbreit [38], Le Métayer [25], and Rosendahl [32]. The idea of simultaneously extracting information about cost and size, and defining the size of a function to be a function itself (leading to higher-order recurrences) has its roots in Danner and Royer [11], which in turn draws from ideas in Shultis [34], Sands [33], and Van Stone [36]. Using bounded modal operators to describe resource usage goes back at least to Girard et al. [13], and the technique has been used fruitfully by countless researchers such as Orchard et al. in the Granule language [30], and Dal Lago and Gaboardi for  $\text{d}\ell\text{PCF}$  [?]. But perhaps the work that is closest in spirit to ours is Benzinger’s ACA system for analyzing call-by-name NUPRL programs [6]. From a cost-annotated operational semantics, he extracts a “symbolic semantics” that is similar in flavor to our recurrence language and extracted recurrences, although without amortization. The symbolic semantics yields higher-order recurrences, which he reduces to first-order recurrences that can be analyzed with a computer algebra system. All of this work focuses on upper bounds for worst-case cost and does not address settings in which tight bounds for a composition of functions are not obtained by composing the corresponding bounds.

There is also extensive work on recurrence extraction from first-order imperative languages. The COSTA project [1–3] takes Java bytecode as its source language, extracts cost relations (essentially, non-deterministic cost recurrences), and solves them for upper bounds. In this line of work, Alonso-Blas and Genaim [4] and Flores-Montoya [12] investigate the failure to derive tight upper bounds in settings where amortized analysis is typically deployed. They trace the issue to the fact that typically cost relations do not depend on the results of the analyzed functions, and hence the computation of bounds cannot make use of such information. Making this possible allows more precise constraints which, when solved, yield tighter bounds. The dependency on output corresponds roughly to total accumulated savings (in the nomenclature of the physicist’s approach to amortized analysis, the potential of a value), and they infer an appropriate potential function, modulo a choice of templates that are used to guide the inference process. To analogize with our work, they delay the determination of the credit policy until solving for upper bounds of extracted recurrences, whereas we specify the credit policy as part of the source program, which directly yields a recurrence for cost that takes the policy into account.

At first glance, the recent work that seems most related to ours is automatic amortized resource analysis (AARA) [14–16, 28], which provides a type inference system for resource bound analysis of higher-order functional programs that incorporates amortization. Credit allocation is built into the type system itself in such a way that unused credits can be used in other parts of the typing. Soundness says that the net credit change during evaluation is bounded by the net credit change described by the typing. AARA focuses primarily on strict languages, but Jost et al. [21] use similar ideas to analyze programs under lazy evaluation, where amortization seems necessary for almost any analysis (see, e.g., Okasaki [29]). To summarize the technical difference between our approach and that of AARA, for us a typing judgment  $\Gamma \vdash_f \cdot : A$  identifies credit usage and the term determines credit allocation (via *save*, *create*, *spend*, etc.), whereas in AARA all of this information is in the type judgment. One place this difference in detail plays out is where and how automation may play a role. For AARA, automation arises via a constraint-based type inference algorithm, which determines the credit allocation strategy, and as implemented, finds bounds if they can be defined in terms of a set of base functions for describing the strategy. For the approach described here, automation could arise in the solution of the extracted recurrences, which does not *a priori* depend on a pre-defined set of base functions. Solving these recurrences is a project that requires future work. It could also be that defining the strategy in the intermediate language, and then reasoning about that strategy (e.g., establishing a credit invariant) in the semantics may provide more flexibility, though again that requires more investigation.

## 7 FUTURE WORK

We expect that the techniques used in [22] to handle general recursion in the source language can be adapted to the approach we have taken here to handle amortization, though work remains to be done to see whether typical non-structurally recursive amortized algorithms would satisfy the necessary typing constraints. A useful project would then be to do the analyses that Okasaki [29] describes via recurrence extraction, which focus on amortized cost of sequences of arbitrary data structure operations (e.g., typical usage of a functional queue), where the data structure is used ephemerally. Adding a type for memoizing thunks to the source language, or more generally lazy evaluation and datatypes, would permit analysis of persistent usage.

We have neglected any discussion of automating either the front end of this process (annotating a program with the constructs used for amortization) or the back end (automatic solving of recurrences). Fully automating the annotation of a source program may be too much to ask (amortized analysis is hard!), but one could hope for a process that elaborates a program with higher-level annotations (e.g. written as comments) into  $\lambda^A$ , inserting *create*, *spend*, etc. On the back end, the



syntactic inequality judgment from Figure 7 can be used to simplify recurrences in  $\lambda^{\mathbb{C}}$  as opposed to interpreting into a model of the recurrence language and then simplifying there. Ideally, one could add enough rules to the judgment (and perhaps enrich its structure) to be able to simplify a large class of standard recurrences, and then apply proof search techniques to automate the process. We would still have higher-order recurrences, and it would be worthwhile to see if the techniques used by Benzinger [6] can be used to reduce them to first-order recurrences that could be solved by a recurrence solver such as OCRS [23].

## REFERENCES

- [1] Elvira Albert, Puri Arenas, Samir Genaim, and Germán Puebla. 2011. Closed-form upper bounds in static cost analysis. *Journal of Automated Reasoning* 46 (2011), 161–203. Issue 2. <https://doi.org/10.1007/s10817-010-9174-1>
- [2] Elvira Albert, Puri Arenas, Samir Genaim, German Puebla, and Damiano Zanardini. 2012. Cost analysis of object-oriented bytecode programs. *Theoretical Computer Science* 413, 1 (2012), 142–159. <https://doi.org/10.1016/j.tcs.2011.07.009>
- [3] Elvira Albert, Samir Genaim, and Abu Naser Masud. 2013. On the Inference of Resource Usage Upper and Lower Bounds. *ACM Transactions on Computational Logic* 14, 3 (2013), 22:1–22:35. <https://doi.org/10.1145/2499937.2499943>
- [4] Diego Esteban Alonso-Blas and Samir Genaim. 2012. On the limits of the classical approach to cost analysis. In *Static Analysis (Lecture Notes in Computer Science)*, Antoine Miné and David Schmidt (Eds.), Vol. 7460. Springer Berlin Heidelberg, 405–421. [https://doi.org/10.1007/978-3-642-33125-1\\_27](https://doi.org/10.1007/978-3-642-33125-1_27)
- [5] Robert Atkey. 2018. Syntax and semantics of quantitative type theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018* (Oxford, United Kingdom). ACM Press, 56–65. <https://doi.org/10.1145/3209108.3209189>
- [6] Ralph Benzinger. 2004. Automated higher-order complexity analysis. *Theoretical Computer Science* 318, 1-2 (2004), 79–103. <https://doi.org/10.1016/j.tcs.2003.10.022>
- [7] Nils Anders Danielsson. 2008. Lightweight semiformal time complexity analysis for purely functional data structures. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008*, George Necula and Philip Wadler (Eds.). ACM Press, 133–144. <https://doi.org/10.1145/1328438.1328457>
- [8] Norman Danner and Daniel R. Licata. 2020. Denotational semantics as a foundation for cost recurrence extraction for functional languages. [arXiv:2002.07262v1](https://arxiv.org/abs/2002.07262v1)
- [9] Norman Danner, Daniel R. Licata, and Ramyaa Ramyaa. 2015. Denotational cost semantics for functional languages with inductive types. In *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015* (Vancouver, BC, Canada), Kathleen Fisher and John Reppy (Eds.). ACM Press, 140–151. <https://doi.org/10.1145/2784731.2784749>
- [10] Norman Danner, Jennifer Paykin, and James S. Royer. 2013. A static cost analysis for a higher-order language. In *Proceedings of the 7th workshop on Programming languages meets program verification, PLPV 2013*, Matthew Might and David Van Horn (Eds.). ACM Press, 25–34. <https://doi.org/10.1145/2428116.2428123>
- [11] Norman Danner and James S. Royer. 2007. Adventures in time and space. *Logical Methods in Computer Science* 3, 9 (2007), 1–53. [https://doi.org/10.2168/LMCS-3\(1:9\)2007](https://doi.org/10.2168/LMCS-3(1:9)2007)
- [12] Antonio Flores-Montoya. 2016. Upper and Lower Amortized Cost Bounds of Programs Expressed as Cost Relations. In *FM 2016: Formal Methods (Lecture Notes in Computer Science)*, John Fitzgerald, Constance Heitmeyer, Stefania Gnesi, and Anna Philippou (Eds.), Vol. 9995. Springer International Publishing, 254–273. [https://doi.org/10.1007/978-3-319-48989-6\\_16](https://doi.org/10.1007/978-3-319-48989-6_16)
- [13] Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. 1992. Bounded linear logic: a modular approach to polynomial-time computability. *Theoretical Computer Science* 97, 1 (1992), 1–66. [https://doi.org/10.1016/0304-3975\(92\)90386-T](https://doi.org/10.1016/0304-3975(92)90386-T)
- [14] Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. 2012. Multivariate Amortized Resource Analysis. *ACM Transactions on Programming Languages and Systems* 34, 3 (2012), 14:1–14:62. <https://doi.org/10.1145/2362389.2362393>
- [15] Jan Hoffmann, Ankush Das, and Shu-Chun Weng. 2017. Towards automatic resource bound analysis for OCaml. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM Press, 359–373. <https://doi.org/10.1145/3009837.3009842>
- [16] Jan Hoffmann and Zhong Shao. 2015. Automatic static cost analysis for parallel programs. In *Programming Languages and Systems: 24th European Symposium on Programming, ESOP 2015 (Lecture Notes in Computer Science)*, Jan Vitek (Ed.), Vol. 9032. Springer-Verlag, 132–157. [https://doi.org/10.1007/978-3-662-46669-8\\_6](https://doi.org/10.1007/978-3-662-46669-8_6)
- [17] Martin Hofmann. 2002. The Strength of Non-Size Increasing Computation. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 260–269. <https://doi.org/10.1145/503272.503297>



- [18] Martin Hofmann. 2003. Linear Types and Non-Size-Increasing Polynomial Time Computation. *Information and Computation* 183, 1 (2003), 57–85. [https://doi.org/10.1016/S0890-5401\(03\)00009-9](https://doi.org/10.1016/S0890-5401(03)00009-9)
- [19] Martin Hofmann and Steffen Jost. 2003. Static prediction of heap space usage for first-order functional programs. In *Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Alex Aiken and Greg Morrisett (Eds.). ACM Press, 185–197. <https://doi.org/10.1145/604131.604148>
- [20] Hudson, Bowornmet. 2016. *Computer-Checked Recurrence Extraction for Functional Programs*. Master’s thesis. Wesleyan University.
- [21] Steffen Jost, Pedro Vasconcelos, Mário Florido, and Kevin Hammond. 2017. Type-based cost analysis for lazy functional languages. *Journal of Automated Reasoning* 59, 1 (2017), 87–120. <https://doi.org/10.1007/s10817-016-9398-9>
- [22] G. A. Kavvos, Edward Morehouse, Daniel R. Licata, and Norman Danner. 2019. Recurrence Extraction for Functional Programs through Call-by-Push-Value. *Proceedings of the ACM on Programming Languages* 4, POPL, Article 15 (2019). <https://doi.org/10.1145/3371083>
- [23] Zachary Kincaid, John Cyphert, Jason Breck, and Thomas Reps. 2017. Non-linear Reasoning for Invariant Synthesis. *Proceedings of the ACM on Programming Languages* 2, POPL, Article 54 (Dec. 2017), 33 pages. <https://doi.org/10.1145/3158142>
- [24] Tristan Knoth, Di Wang, Nadia Polikarpova, and Jan Hoffmann. 2019. Resource-Guided Program Synthesis. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*.
- [25] Daniel Le Métayer. 1988. ACE: an automatic complexity evaluator. *ACM Transactions on Programming Languages and Systems* 10, 2 (1988), 248–266. <https://doi.org/10.1145/42190.42347>
- [26] Daniel R. Licata, Michael Shulman, and Mitchell Riley. 2017. A fibrational framework for substructural and modal logics. In *2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017 (Leibniz International Proceedings in Informatics (LIPIcs))*, Dale Miller (Ed.), Vol. 84. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 25:1–25:22. <https://doi.org/10.4230/LIPIcs.FSCD.2017.25>
- [27] Conor McBride. 2016. I got plenty o’ Nuttin’. In *A List of Successes That Can Change the World: Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday (Lecture Notes in Computer Science)*, Sam Lindley, Conor McBride, Phil Trinder, and Don Sannella (Eds.), Vol. 9600. Springer-Verlag. [https://doi.org/10.1007/978-3-319-30936-1\\_12](https://doi.org/10.1007/978-3-319-30936-1_12)
- [28] Yue Niu and Jan Hoffmann. 2018. Automatic Space Bound Analysis for Functional Programs with Garbage Collection. In *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning (EPIc Series in Computing)*, Gilles Barthe, Geoff Sutcliffe, and Margus Veanes (Eds.), Vol. 57. EasyChair, 543–563. <https://doi.org/10.29007/xkwx>
- [29] Chris Okasaki. 1998. *Purely Functional Data Structures*. Cambridge University Press.
- [30] Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. 2019. Quantitative program reasoning with graded modal types. *Proceedings of the ACM on Programming Languages* 3, ICFP (2019), 110:1–110:30. <https://doi.org/10.1145/3341714>
- [31] Jason Reed. [n.d.]. Names are (mostly) useless. ([n.d.]). <https://www.cis.upenn.edu/~sweirich/wmm/wmm08-programme.html> Presented at 3rd Informal ACM SIGPLAN Workshop on Mechanizing Metatheory 2008.
- [32] Mads Rosendahl. 1989. Automatic complexity analysis. In *Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture, FPCA 1989*, Joseph E. Stoy (Ed.). ACM Press, 144–156. <https://doi.org/10.1145/99370.99381>
- [33] David Sands. 1990. *Calculi for Time Analysis of Functional Programs*. Ph.D. Dissertation. University of London.
- [34] Jon Shultis. 1985. *On the complexity of higher-order programs*. Technical Report CU-CS-288-85. University of Colorado at Boulder.
- [35] Daniel Dominic Sleator and Robert Endre Tarjan. 1985. Self-adjusting binary search trees. *Journal of the ACM* 32, 3 (1985).
- [36] Kathryn Van Stone. 2003. *A Denotational Approach to Measuring Complexity in Functional Programs*. Ph.D. Dissertation. Carnegie Mellon University.
- [37] Robert Endre Tarjan. 1985. Amortized computational complexity. *SIAM J. Algebraic Discrete Methods* 6, 2 (1985), 306–318. <https://doi.org/10.1137/0606031>
- [38] Ben Wegbreit. 1975. Mechanical program analysis. *Communications of the Association for Computing Machinery* 18, 9 (1975), 528–539. <https://doi.org/10.1145/361002.361016>

## A APPENDIX

THEOREM 2.1 (ADMISSIBLE STRUCTURAL RULES).

- *Resource Weakening*: Write  $g \geq f$  for the coefficient-wise partial order on resource terms ( $a_1x_1 + a_2x_2 + \dots + \ell \geq b_1x_1 + b_2x_2 + \dots + \ell'$  iff  $a_i \geq b_i$  for all  $i$  and  $\ell \geq \ell'$ ). Then if  $\Gamma \vdash_f M : A$  and  $g \geq f$  then  $\Gamma \vdash_g M : A$ .
- *Variable Weakening*: If  $\Gamma \vdash_f M : A$  and  $y$  does not occur in  $\Gamma$ , then  $\Gamma, y : B \vdash_{f+0y} M : A$ .
- *Substitution*: If  $\Gamma \vdash_f M : A$  and  $\Gamma, x : A \vdash_g N : B$ , then  $\Gamma \vdash_{g[f/x]} N[M/x] : B$

PROOF. All follow by straightforward induction on judgments.  $\square$

THEOREM 2.2 (FUSION LAWS).

- (1)  $!_{\ell_1+k_1}^{k_1k_2} A \dashv\vdash !_{\ell_1}^{k_1} !_{\ell_2}^{k_2} A$
- (2)  $!_{\ell_1+\ell_2}^k (A \otimes B) \dashv\vdash !_{\ell_1}^k A \otimes !_{\ell_2}^k B$
- (3)  $!_{\ell}^k (A \oplus B) \dashv\vdash !_{\ell}^k A \oplus !_{\ell}^k B$

PROOF. We present terms going in both directions for each caes.

- (1)  $x : !_{\ell_1+k_1}^{k_1k_2} \vdash_x \text{transfer}_1 !_{\ell_1+k_1}^{k_1k_2} y = x$  to  $\text{save}_{\ell_1}^{k_1} (\text{save}_{\ell_2}^{k_2} y) : !_{\ell_1}^{k_1} !_{\ell_2}^{k_2} A$  and  $x : !_{\ell_1}^{k_1} !_{\ell_2}^{k_2} A \vdash_x \text{transfer}_1 !_{\ell_1}^{k_1} y = x$  to  $\text{transfer}_{k_1} !_{\ell_2}^{k_2} z = y$  to  $\text{save}_{\ell_1+k_1}^{k_1k_2} z : !_{\ell_1+k_1}^{k_1k_2} A$
- (2)  $x : !_{\ell_1+\ell_2}^k (A \otimes B) \vdash_x \text{transfer}_1 !_{\ell_1+\ell_2}^k y = x$  to  $\text{split}_{k'}(y, z_1.z_2.(\text{save}_{\ell_1}^k z_1, \text{save}_{\ell_2}^k z_2)) : !_{\ell_1}^k A \otimes !_{\ell_2}^k B$  and  $x : !_{\ell_1}^k A \otimes !_{\ell_2}^k B \vdash_x \text{split}_1(x, z_1.z_2.\text{transfer}_1 !_{\ell_1}^k y_1 = z_1$  to  $\text{transfer}_1 !_{\ell_2}^k y_2 = z_2$  to  $\text{save}_{\ell_1+\ell_2}^k (y_1, y_2)) : !_{\ell_1+\ell_2}^k (A \otimes B)$ .
- (3)  $x : !_{\ell}^k (A \oplus B) \vdash_x \text{transfer}_1 !_{\ell}^k y = x$  to  $\text{case}_k(y, z_1.\text{inl}(\text{save}_{\ell}^k z_1), z_2.\text{inr}(\text{save}_{\ell}^k z_2)) : !_{\ell}^k A \oplus !_{\ell}^k B$  and  $x : !_{\ell}^k A \oplus !_{\ell}^k B \vdash_x \text{case}_1(x, z_1.\text{transfer}_1 !_{\ell}^k y = z_1$  to  $\text{save}_{\ell}^k (\text{inl } y), z_2.\text{transfer}_1 !_{\ell}^k y = z_2$  to  $\text{save}_{\ell}^k (\text{inr } y)) : !_{\ell}^k (A \oplus B)$

$\square$

THEOREM 2.3 (PRESERVATION BOUND). If  $\cdot \vdash_a M : A$  and  $M \downarrow^{(n,r)} v$ , then  $a+r \geq 0$  and  $\cdot \vdash_{a+r} v : A$ .

PROOF. By induction on  $M \downarrow v$ .

- (Values): Suppose  $\cdot \vdash_a v : A$  and  $v \downarrow^{(0,0)} v$ . Then,  $a+0 \geq 0$  (because  $a \geq 0$ ), and  $\cdot \vdash_{a+0=a} v : A$ .
- (Tick): Immediate by IH.
- (!-I): Suppose  $\cdot \vdash_b \text{save}_l^k M : !_l^k A$ , and  $\text{save}_l^k M \downarrow^{(\cdot, kr)} \text{save}_l^k v$ . We must show that  $b+kr \geq 0$  and that  $\cdot \vdash_{b+kr} \text{save}_l^k v : !_l^k A$ . Inverting the rules, we have that  $\cdot \vdash_a M : A$  with  $ka+l \leq b$ , and that  $M \downarrow^{(n,r)} v$ . By IH,  $\cdot \vdash_{a+r} v : A$  with  $a+r \geq 0$ . Since  $k, l \geq 0, 0 \leq k(a+r)+l = ka+l+kr$ , which, since  $ka+l \leq b$ , is less than or equal to  $b+kr$ . So,  $b+kr \geq 0$  and  $\cdot \vdash_{b+kr} \text{save}_l^k v : !_l^k A$ , as required.
- (!-E): For this case, suppose  $\cdot \vdash_{k'a+b} \text{transfer}_{k'} !_l^k x = M$  to  $N : C$ , and  $\text{transfer}_{k'} !_l^k x = M$  to  $N \downarrow^{(\cdot, k'r_1+r_2)} v$ . We want to show that:  $k'a+b+k'r_1+r_2 \geq 0$  and  $\cdot \vdash_{k'a+b+k'r_1+r_2} v : C$ . By inversion,  $\cdot \vdash_a M : !_l^k A$ , and  $x : A \vdash_{b+k'(kx+l)} N : C$ , as well as  $M \downarrow^{(\cdot, r_1)} \text{save}_l^k v_1$  and  $N[v_1/x] \downarrow^{(\cdot, r_2)} v$ . By IH, we know that  $\cdot \vdash_{a+r_1} \text{save}_l^k v_1 : !_l^k A$  and  $a+r_1 \geq 0$ , so by inversion, there is a  $d$  such that  $kd+l \leq a+r_1$ , and  $\cdot \vdash_d v_1 : A$ , and so substitution gives that  $\cdot \vdash_{b+k'(kd+l)} N[v_1/x] : C$ . But  $kd+l \leq a+r_1$ , so by structural weakening we have  $\cdot \vdash_{b+k'a+k'r_1} N[v_1/x] : C$ . Again by IH,  $\cdot \vdash_{b+r_2+k'a+k'r_1} v : C$  and  $b+r_2+k'a+k'r_1 \geq 0$ , as required.
- (create): Suppose  $\cdot \vdash_a \text{create}_l M : A$  and  $\text{create}_l M \downarrow^{(n,r+l)} v$ . Inverting, we have  $\cdot \vdash_{a+l} M : A$ , and  $M \downarrow^{(n,r)} v$ . By IH, we have that  $a+l+r \geq 0$ , and  $\cdot \vdash_{a+r+l} v$ . But, we wanted to show that  $a+r+l \geq 0$  and that  $\cdot \vdash_{a+r+l} v : A$ , and so we are done.

- (spend): Suppose  $\cdot \vdash_{a+l} \text{spend}_l M : A$ , and  $\text{spend}_l M \downarrow^{(\cdot, r-l)} v$ . We want to show that  $a + l + r - l = a + r \geq 0$ , and that  $\cdot \vdash_{a+l+r-l=a+r} v$  Inverting, we have that  $\cdot \vdash_a M : A$  and  $M \downarrow^{(n, r)} v$ . By IH,  $a + r \geq 0$  and  $\cdot \vdash_{a+r} v : A$ , as required.
- ( $\otimes$ -I): For this case, let  $\cdot \vdash_{a+b} (M, N) : A \otimes B$ , and  $(M, N) \downarrow^{(\cdot, k_1+k_2)} (v_1, v_2)$ . We must show that  $a + b + k_1 + k_2 \geq 0$ , and that  $\cdot \vdash_{a+b+k_1+k_2} (v_1, v_2)$  Inverting, we get the four premises  $\cdot \vdash_a M : A$ ,  $\cdot \vdash_b N : B$ , and  $M \downarrow^{(\cdot, k_1)} v_1$  and  $N \downarrow^{(\cdot, k_2)} v_2$ . Using the IH on these two pairs, we get that  $a + k_1 \geq 0$ ,  $b + k_2 \geq 0$ ,  $\cdot \vdash_{a+k_1} v_1 : A$ , and  $\cdot \vdash_{b+k_2} v_2 : B$ . Adding the two inequalities and applying  $\otimes$ -I to the judgments gives the desired result.
- ( $\oplus$ -E): Suppose  $\cdot \vdash_{a+b_1+b_2} \text{case}_{k'} (M, x.N_1, y.N_2) : C$  and  $\text{case}_{k'} (M, x.N_1, y.N_2) \downarrow^{(\cdot, k'r_1+r_2)} v$ . Inverting the typing judgment,  $\cdot \vdash_a M : A \oplus B$ ,  $x : A \vdash_{b_1+k'x} N_1 : C$  and  $y : B \vdash_{b_2+k'y} N_2 : C$ . Inverting the evaluation judgment gives two symmetric cases, so suppose that  $M \downarrow^{(\cdot, r_1)} \text{inl } v_1$  and  $N_1[v_1/x] \downarrow^{(\cdot, r_2)} v$ . By IH,  $\cdot \vdash_{a+r_1} \text{inl } v_1 : A \oplus B$  and  $a + r_1 \geq 0$ . So,  $\cdot \vdash_{a+r_1} v_1 : A$ . By substitution,  $\cdot \vdash_{b_1+k'(a+r_1)} N_1[v_1/x] : C$ . By IH,  $\cdot \vdash_{k'a+b_1+k'r_1+r_2} v : C$  and  $k'a+b_1+k'r_1+r_2 \geq 0$ . Since  $b_2 \geq 0$ , by structural weakening,  $\cdot \vdash_{k'a+b_1+b_2+k'r_1+r_2} v : C$  and  $k'a+b_1+b_2+k'r_1+r_2 \geq 0$ , as required.
- ( $\multimap$ -E): Let  $\cdot \vdash_{a+b} MN : B$ , and  $MN \downarrow^{(\cdot, k_1+k_2+k_3)} v$ . We want to show that  $a+b+k_1+k_2+k_3 \geq 0$ , and that  $\cdot \vdash_{a+b+k_1+k_2+k_3} v : B$ . We invert both judgments to get  $\cdot \vdash_a M : A \multimap B$ , and  $\cdot \vdash_b N : A$ , and  $M \downarrow^{(\cdot, k_1)} \lambda x.M'$ , and  $N \downarrow^{(\cdot, k_2)} v_1$ , and that  $M'[v_1/x] \downarrow^{(\cdot, k_3)} v$ . Applying the IH to the first evaluation, we have that  $\cdot \vdash_{a+k_1} \lambda x.M' : A \multimap B$ . Inverting the proof of that judgment, we get that  $x : A \vdash_{a+k_1+x} M' : B$ . By IH again,  $\cdot \vdash_{b+k_2} v_1 : A$ , and by substitution,  $\cdot \vdash_{a+b+k_1+k_2} M'[v_1/x] : B$ . By IH once more,  $a + b + k_1 + k_2 + k_3 \geq 0$ , and  $\cdot \vdash_{a+b+k_1+k_2+k_3} v : B$ , as required.
- ( $\mathbb{N}$ -E) Suppose  $\cdot \vdash_{a+b_1+b_2} \text{nrec} (M, N_1, N_2) : C$ . By inversion,  $\cdot \vdash_a M : \mathbb{N}$ ,  $\cdot \vdash_{b_1} N_1 : 1 \multimap C$ , and  $\cdot \vdash_{b_2} N_2 : !_0^\infty (\mathbb{N} \otimes (1 \multimap C) \multimap C)$ . We have two evaluation cases to consider.
  - Suppose  $\text{nrec} (M, N_1, N_2) \downarrow^{(\cdot, r_1+r_2+r_3+r_4)} v$  by way of  $M \downarrow^{(\cdot, r_1)} 0 : \mathbb{N}$ ,  $N_1 \downarrow^{(\cdot, r_2)} \lambda x.N'_1$ ,  $N_2 \downarrow^{(\cdot, r_3)} \_$ , and  $N'_1[(\cdot)/x] \downarrow^{(\cdot, r_4)} v$ . Then, by IH, we have the following:
    - \*  $\cdot \vdash_{a+r_1} 0 : \mathbb{N}$ , and  $a + r_1 \geq 0$
    - \*  $\cdot \vdash_{b_1+r_2} \lambda x.N'_1 : 1 \multimap C$ ,  $b_1 + r_2 \geq 0$ .
    - \*  $b_2 + r_3 \geq 0$
 Since  $\cdot \vdash_0 (\cdot) : 1$ ,  $\cdot \vdash_{b_1+r_1} N'_1[(\cdot)/x] : C$ . By IH,  $\cdot \vdash_{b_1+r_2+r_4} v : C$ . By structural weakening,  $\cdot \vdash_{a+b_1+b_2+r_1+r_2+r_3+r_4} v : C$ , as required.
  - Suppose  $\text{nrec} (M, N_1, N_2) \downarrow^{(\cdot, r_1+r_2+r_3+r_4)} v$  by way of  $M \downarrow^{(\cdot, r_1)} S(v_1) : \mathbb{N}$ ,  $N_1 \downarrow^{(\cdot, r_2)} \lambda x.N'_1$ ,  $N_2 \downarrow^{(\cdot, r_3)} \text{save}_0^\infty (\lambda x.N'_2)$ , and  $N'_2[(v_1, \lambda z.\text{nrec} (v_1, \lambda x.N'_1, \text{save}_0^\infty (\lambda x.N'_2)))/x] \downarrow^{(\cdot, r_4)} v$ . Then, by IH we have:
    - \*  $\cdot \vdash_{a+r_1} S(v_1) : \mathbb{N}$ ,  $a + r_1 \geq 0$
    - \*  $\cdot \vdash_{b_1+r_2} \lambda x.N'_1 : 1 \multimap C$ ,  $b_1 + r_2 \geq 0$
    - \*  $\cdot \vdash_{b_2+r_3} \text{save}_0^\infty (\lambda x.N'_2) : !_0^\infty (\mathbb{N} \otimes (1 \multimap C) \multimap C)$ , and  $b_2 + r_3 \geq 0$ .
 By  $\mathbb{N}$ -strengthening,  $\cdot \vdash_0 S(v_1) : \mathbb{N}$ , and so  $\cdot \vdash_0 v_1 : \mathbb{N}$ . Since  $\cdot \vdash_{b_2+r_3} \text{save}_0^\infty (\lambda x.N'_2) : !_0^\infty (\mathbb{N} \otimes (1 \multimap C) \multimap C)$ , there is a  $c \geq 0$  so that  $\infty \cdot c \leq b_2 + r_3$  with  $\cdot \vdash_c \lambda x.N'_2 : \mathbb{N} \otimes (1 \multimap C) \multimap C$ . Then,  $\cdot \vdash_{b_1+r_3+\infty \cdot c} \text{nrec} (v_1, \lambda x.N'_1, \text{save}_0^\infty (\lambda x.N'_2)) : C$ , and so  $\cdot \vdash_{a+b_1+r_1+r_2+\infty \cdot c} (v_1, \lambda z.\text{nrec} (v_1, \lambda x.N'_1, \text{save}_0^\infty (\lambda x.N'_2))) : \mathbb{N} \otimes (1 \multimap C)$ . Thus, since  $x : \mathbb{N} \otimes (1 \multimap C) \vdash_{x+c} N'_2 : C$ ,
 
$$\cdot \vdash_{a+b_1+r_1+r_2+\infty \cdot c} N'_2[(v_1, \lambda z.\text{nrec} (v_1, \lambda x.N'_1, \text{save}_0^\infty (\lambda x.N'_2)))/x] : C$$

since  $\infty \cdot c + c = \infty \cdot c$ . So, by IH,  $\cdot \vdash_{a+b_1+r_1+r_2+\infty \cdot c+r_4} v : C$ , and because  $\infty \cdot c \leq b_2 + r_3$ , we have by weakening that  $\cdot \vdash_{a+b_1+b_2+r_1+r_2+r_3+r_4} v : C$  as required.

- ([A]-E) Suppose  $\cdot \vdash_{a+b_1+b_2} \text{lrec}(M, N_1, N_2) : C$ . Then,  $\cdot \vdash_a M : A$ ,  $\cdot \vdash_{b_1} N_1 : 1 \rightarrow C$ , and  $\cdot \vdash_{b_2} N_2 : !_0^\infty(A \otimes (\text{List}(A) \& C) \multimap C)$ . We have two evaluation cases to consider.  
 Firstly, suppose  $\text{lrec}(M, N_1, N_2) \downarrow^{(\cdot, r_1+r_2+r_3+r_4)} v$  by way of  $M \downarrow^{(\cdot, r_1)} v$ ,  $N_1 \downarrow^{(\cdot, r_2)} \lambda x. N'_1$ ,  $N_2 \downarrow^{(\cdot, r_3)} v'$ , and  $N'_1[(\cdot)/x] \downarrow^{(\cdot, r_4)} v$ . Then, by IH,  $a + r_1 \geq 0$ , which means that  $\cdot \vdash_{a+r_1} (\cdot) : 1$ . By IH,  $\cdot \vdash_{b_1+r_2} \lambda x. N'_1$  and  $b_1 + r_2 \geq 0$ . So, by inversion and then substitution,  $\cdot \vdash_{a+b_1+r_1+r_2} N'_1[(\cdot)/x] : C$ . By IH,  $b_2 + r_3 \geq 0$ , so by weakening,  $\cdot \vdash_{a+b_1+b_2+r_1+r_2+r_3} N'_1[(\cdot)/x]$ . Finally, by IH,  $a + b_1 + b_2 + r_1 + r_2 + r_3 + r_4 \geq 0$  and  $\cdot \vdash_{a+b_1+b_2+r_1+r_2+r_3+r_4} v : C$ .  
 Now, suppose  $\text{lrec}(M, N_1, N_2) \downarrow^{(\cdot, r_1+r_2+r_3+r_4)} v$  by way of  $M \downarrow^{(\cdot, r_1)} v_1 :: v_2$ ,  $N_1 \downarrow^{(\cdot, r_2)} \lambda x. N'_1$ ,  $N_2 \downarrow^{(\cdot, r_3)} \text{save}_0^\infty(\lambda x. N'_2)$ , and  $N'_2[(v_1, \langle v_2, \text{lrec}(v_2, \lambda x. N'_1, \text{save}_0^\infty(\lambda x. N'_2)) \rangle)/x] \downarrow^{(\cdot, r_4)} v$ . By IH,  $\cdot \vdash_{a+r_1} v_1 :: v_2 : \text{List}(A)$  and  $a + r_1 \geq 0$ . By inversion, there are  $d_1, d_2 \geq 0$  so that  $a + r_1 = d_1 + d_2$  and  $\cdot \vdash_{d_1} v_1 : A$  and  $\cdot \vdash_{d_2} v_2 : \text{List}(A)$ . By two more applications of the IH,  $\cdot \vdash_{b_1+r_2} \lambda x. N'_1 : 1 \multimap C$ ,  $\cdot \vdash_{b_2+r_3} \text{save}_0^\infty(\lambda x. N'_2) : !_0^\infty(A \otimes (\text{List}(A) \& C) \multimap C)$ , with  $b_1 + r_2 \geq 0$  and  $b_2 + r_3 \geq 0$ . By inversion, there is some  $c \geq 0$  with  $\infty \cdot c \leq b_2 + r_3$  such that  $\cdot \vdash_c \lambda x. N'_2 : A \otimes (\text{List}(A) \& C) \multimap C$ . Next,  

$$\frac{\cdot \vdash_{d_2} v_2 : \text{List}(A) \quad \cdot \vdash_{b_1+r_2} \lambda x. N'_1 : 1 \multimap C \quad \cdot \vdash_{\infty \cdot c} \text{save}_0^\infty(\lambda x. N'_2) : !_0^\infty(A \otimes (\text{List}(A) \& C) \multimap C)}{\cdot \vdash_{d_2+b_1+r_2+\infty \cdot c} \text{lrec}(v_2, \lambda x. N'_1, \text{save}_0^\infty(\lambda x. N'_2)) : C}$$
 then, with  $\cdot \vdash_{d_2+b_1+r_2+\infty \cdot c} v_2 : \text{List}(A)$ , we have that  

$$\cdot \vdash_{d_2+b_1+r_2+\infty \cdot c} \langle v_2, \text{lrec}(v_2, \lambda x. N'_1, \text{save}_0^\infty(\lambda x. N'_2)) \rangle : \text{List}(A) \& C$$
 and since  $\cdot \vdash_{d_1} v_1 : A$ ,  

$$\cdot \vdash_{a+r_1+b_1+r_2+\infty \cdot c} (v_1, \langle v_2, \text{lrec}(v_2, \lambda x. N'_1, \text{save}_0^\infty(\lambda x. N'_2)) \rangle) : A \otimes (\text{List}(A) \& C)$$
 and so by substitution, and using the fact that  $c + \infty \cdot c = \infty \cdot c$ ,  $\cdot \vdash_{a+b_1+r_1+r_2+\infty \cdot c} N'_2[(v_1, \langle v_2, \text{lrec}(v_2, \lambda x. N'_1, \text{save}_0^\infty(\lambda x. N'_2)) \rangle)/x] : C$ . By weakening, since  $\infty \cdot c \leq b_2 + r_3$ ,  $\cdot \vdash_{a+b_1+b_2+r_1+r_2+r_3} N'_2[(v_1, \langle v_2, \text{lrec}(v_2, \lambda x. N'_1, \text{save}_0^\infty(\lambda x. N'_2)) \rangle)/x] : C$ . Finally, by IH,  $\cdot \vdash_{a+b_1+b_2+r_1+r_2+r_3+r_4} v : C$ , and  $a + b_1 + b_2 + r_1 + r_2 + r_3 + r_4 \geq 0$ , as required.
- (&-I): Immediate.
- (&-E): By symmetry, it suffices to only consider the  $\pi_1$  case. Let  $\cdot \vdash_a \pi_1 M : A$  and  $\pi_1 M \downarrow^{(\cdot, r_1+r_2)} v$ . By inversion, we have that  $\cdot \vdash_a M : A \& B$ , and that  $M \downarrow^{(\cdot, r_1)} \langle N_1, N_2 \rangle$  and  $N_1 \downarrow^{(\cdot, r_2)}$ . We must show that  $\cdot \vdash_{a+r_1+r_2} v : A$ , and that  $a + r_1 + r_2 \geq 0$ . By IH,  $\cdot \vdash_{a+r_1} \langle N_1, N_2 \rangle : A \& B$ . Inverting this, we get that  $\cdot \vdash_{a+r_1} N_1 : A$ , and so again by IH,  $\cdot \vdash_{a+r_1+r_2} v : A$ , and  $a + r_1 + r_2 \geq 0$ , as required.

□

THEOREM 2.4. *If  $v$  is a value, and  $v \downarrow^{(n,r)} v$ , then  $n = r = 0$ .*

PROOF. By inspection of cases.

□

THEOREM 2.5 (RESOURCE STRENGTHENING FOR  $\mathbb{N}$ ). *If  $\cdot \vdash_a v : \mathbb{N}$ , then  $\cdot \vdash_0 v : \mathbb{N}$*

PROOF. By canonical forms,  $v = \bar{n}$ , proceed by induction on  $n$ .

□

THEOREM 3.1 (EXTRACTION PRESERVES TYPES). *If  $\Gamma \vdash_a M : A$  then  $\langle\langle \Gamma \rangle\rangle \vdash \|M\| : \|A\|$*

PROOF. By induction on  $\Gamma \vdash_f M : A$

□

THEOREM 3.4 (WEAKENING).

- (1) *If  $M \sqsubseteq^{A,a} E$ , and  $E \leq_{\|A\|} E'$ , then  $M \sqsubseteq^{A,a} E'$*
- (2) *If  $v \sqsubseteq_{\text{val}}^{A,a} E$ , and  $E \leq_{\langle\langle A \rangle\rangle} E'$ , then  $v \sqsubseteq_{\text{val}}^{A,a} E'$*

PROOF. We prove 1 and 2 simultaneously by induction on  $A$ .

(1) Suppose  $M \sqsubseteq^{A,a} E$ , and  $E \leq_{\mathbb{C} \times \langle\langle A \rangle\rangle} E'$ . We need to show that  $M \sqsubseteq^{A,a} E'$ . Suppose  $M \downarrow^{(n,r)} v$ .

We need to show:

- $n \leq E'_c - r$
- $v \sqsubseteq_{\text{val}}^{A,a+r} E'_p$

But, since  $M \sqsubseteq^{A,a} E$

- $n \leq E_c - r$
- $v \sqsubseteq_{\text{val}}^{A,a+r} E_p$

so, it suffices to show that  $E_c \leq_{\mathbb{C}} E'_c$  and  $E_p \leq_{\langle\langle A \rangle\rangle} E'_p$ , which is true by the  $\pi_1(-)$  and  $\pi_2(-)$  congruences, recalling that  $(-)_c$  and  $(-)_p$  are simply  $\pi_1$  and  $\pi_2$ .

(2) Let  $E \leq_{\langle\langle A \rangle\rangle} E'$ . We have a few cases to consider.

(!) Suppose  $\text{save}_l^k v \sqsubseteq_{\text{val}}^{!l^k A, a} E$ . We must show that  $\text{save}_l^k v \sqsubseteq_{\text{val}}^{!l^k A, a} E'$ . We know that there is a  $d \geq 0$  such that  $ka + l \leq d$ , and  $v \sqsubseteq_{\text{val}}^{A,d} E$ . So, by IH,  $v \sqsubseteq_{\text{val}}^{A,d} E'$ , and hence  $\text{save}_l^k v \sqsubseteq_{\text{val}}^{!l^k A, a} E'$ , as required.

( $\multimap$ ) Suppose  $\lambda x.M \sqsubseteq_{\text{val}}^{A \multimap B, a} E$ . We need to show that  $\lambda x.M \sqsubseteq_{\text{val}}^{A \multimap B, a} E'$ . Let  $v \sqsubseteq_{\text{val}}^{A,b} E_v$ . Then,  $M[v/x] \sqsubseteq_{\text{val}}^{B, a+b} E E_v$ . Using the application congruence and 1,  $M[v/x] \sqsubseteq_{\text{val}}^{B, a+b} E' E_v$ . Since  $v, b, E_v$  were chosen arbitrarily,  $\lambda x.M \sqsubseteq_{\text{val}}^{A \multimap B, a} E'$  as required.

( $\otimes$ ) Suppose  $(v_1, v_2) \sqsubseteq_{\text{val}}^{A_1 \otimes A_2, a} E$ . Then, there are  $a_1, a_2$  such that  $a_1 + a_2 = a$ , and  $v_i \sqsubseteq_{\text{val}}^{A_i, a_i} \pi_i E$ , and so by  $\pi_i$ -congruence and the IH,  $v_i \sqsubseteq_{\text{val}}^{A_i, a_i} \pi_i E'$ , so  $(v_1, v_2) \sqsubseteq_{\text{val}}^{A_1 \otimes A_2, a} E'$ , as required.

(List (A)) Both cases are immediate by transitivity.

( $\mathbb{N}$ ) Both cases are immediate by transitivity.

( $\oplus$ ) Both cases are immediate by transitivity.

(A&B) Suppose  $\langle M_1, M_2 \rangle \sqsubseteq_{\text{val}}^{A_1 \& A_2, a} E$ . Then, for  $i \in \{1, 2\}$ ,  $M_i \sqsubseteq_{\text{val}}^{A_i, a} \pi_i E$ . By  $\pi_i$ -congruence,  $\pi_i E \leq_{\|A\|} \pi_i E'$ , and so by IH from 1, we know that  $M_i \sqsubseteq_{\text{val}}^{A_i, a} \pi_i E'$ , and are done.

□

THEOREM 3.5 (CREDIT WEAKENING). *If  $a_1 \leq a_2$ , then:*

(1) *If  $M \sqsubseteq^{A, a_1} E$ , then  $M \sqsubseteq^{A, a_2} E$*

(2) *If  $v \sqsubseteq_{\text{val}}^{A, a_1} E$ , then  $v \sqsubseteq_{\text{val}}^{A, a_2} E$*

PROOF. We prove the two claims simultaneously.

(1) Suppose  $M \sqsubseteq^{A, a_1} E$ . To show  $M \sqsubseteq^{A, a_2} E$ , suppose  $M \downarrow^{(n,r)} v$ . We must show that such that

- $n \leq E_c - r$
- $v \sqsubseteq_{\text{val}}^{A, a_2+r} E_p$

But, since  $M \sqsubseteq^{A, a_1} E$ , we have

- $n \leq E_c - r$
- $v \sqsubseteq_{\text{val}}^{A, a_1+r} E_p$

Since  $a_1 \leq a_2$ ,  $a_1 + r \leq a_2 + r$ , so we are done by (2).

(2) By lexicographic induction on first  $A$  and then the size of  $v$ .

(!) Let  $\text{save}_l^k v \sqsubseteq_{\text{val}}^{!l^k A, a_1} E$ . Then, there is a  $d \geq 0$  such that  $kd + l \leq a_1$  and  $v \sqsubseteq_{\text{val}}^{A,d} E$ . But

$$kd + l \leq a_1 \leq a_2, \text{ and so } \text{save}_l^k v \sqsubseteq_{\text{val}}^{!l^k A, a_2} E$$

( $\multimap$ ) Let  $\lambda x.M \sqsubseteq_{\text{val}}^{A \multimap B, a_1} E$ . Suppose  $v \sqsubseteq_{\text{val}}^{A,b} E'$ . Then,  $M[v/x] \sqsubseteq_{\text{val}}^{B, a_1+b} E E'$ , and so by (1),  $M[v/x] \sqsubseteq_{\text{val}}^{B, a_2+b} E E'$ . Since  $v$  was chosen arbitrarily,  $\lambda x.M \sqsubseteq_{\text{val}}^{A \multimap B, a_2} E$ , as required.

- ( $\otimes$ ) Let  $(v_1, v_2) \sqsubseteq_{\text{val}}^{A_1 \otimes A_2, a_1} E$ . Then, there are  $b_1, b_2$  with  $b_1 + b_2$  such that  $b_1 + b_2 = a_1$ , and  $v_i \sqsubseteq_{\text{val}}^{A_i, a_i} \pi_i E$  for  $i \in \{1, 2\}$ . By the IH on  $v_1$ , we have that  $v_1 \sqsubseteq_{\text{val}}^{A_1, b_1 + a_2 - a_1} \pi_1 E$ , and so  $(v_1, v_2) \sqsubseteq_{\text{val}}^{A_1 \otimes A_2, a_2} E$ , as required.
- (List (A)) The empty case is immediate. Suppose  $v_1 :: v_2 \sqsubseteq_{\text{val}}^{\text{List}(A), a_1} E$ . Then, there are  $E_1, E_2, b_1, b_2$  such that  $b_1 + b_2 = a_1$ ,  $E_1 :: E_2 \leq E$ ,  $v_1 \sqsubseteq_{\text{val}}^{A, b_1} E_1$ , and  $v_2 \sqsubseteq_{\text{val}}^{\text{List}(A), b_2} E_2$ . By IH,  $v_1 \sqsubseteq_{\text{val}}^{A, b_1 + a_2 - a_1} E_1$ , and so  $v_1 :: v_2 \sqsubseteq_{\text{val}}^{\text{List}(A), a_2} E$ .
- ( $\mathbb{N}$ ) The zero case is immediate. Suppose  $S(v) \sqsubseteq_{\text{val}}^{\mathbb{N}, a_1} E$ . Then, there is  $E'$  such that  $S(E') \leq E$ , and  $v \sqsubseteq_{\text{val}}^{\mathbb{N}, a_1} E'$ . Since  $v$  is a smaller term than  $S(v)$ , we can apply the IH to see that  $v \sqsubseteq_{\text{val}}^{\mathbb{N}, a_2} E'$ , and so  $S(v) \sqsubseteq_{\text{val}}^{\mathbb{N}, a_2} E$ , as desired.
- ( $\oplus$ ) The two cases are symmetric, so we present only one. Suppose  $\text{inl } v \sqsubseteq_{\text{val}}^{A \oplus B, a_1} E$ . Then we have  $E'$  such that  $\text{inl } E' \leq E$ , and  $v \sqsubseteq_{\text{val}}^{A, a_1} E'$ , which, by IH, means that  $v \sqsubseteq_{\text{val}}^{A, a_2} E'$ , and so  $\text{inl } v \sqsubseteq_{\text{val}}^{A \oplus B, a_2} E$ .
- (A&B) Immediate by IH.

□

**THEOREM 3.6 ( $\mathbb{N}$ -RECURSOR).** *If  $\lambda x.N'_1 \sqsubseteq_{\text{val}}^{1-\circ C, c_3} E_1$ ,  $\lambda x.N'_2 \sqsubseteq_{\text{val}}^{\mathbb{N} \otimes (1-\circ C) \rightarrow \circ C, d} E_2$  with  $d \geq 0$ , then  $\forall n \geq 0$ , if  $\bar{n} \sqsubseteq_{\text{val}}^{\mathbb{N}, 0} E$ , then  $\text{nrec}(\bar{n}, \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)) \sqsubseteq_{\text{val}}^{C, c_3 + \infty \cdot d} \text{nrec}(E, E_1, \lambda p.E_2(\pi_1 p, \lambda z.\pi_2 p))$*

**PROOF.** Proceed by induction on  $n$ .

For notational simplicity, let  $E_2^* = \lambda p.E_2(\pi_1 p, (\lambda z.\pi_2 p))$

- ( $n = 0$ ): To show  $\text{nrec}(0, \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)) \sqsubseteq_{\text{val}}^{C, c_3 + \infty \cdot d} \text{nrec}(E, E_1, E_2^*)$ , suppose that  $\text{nrec}(0, \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)) \downarrow^{(n, r)} v$  by way of  $N'_1[(\cdot)/x] \downarrow^{(n, r)}$ .

We must show that :

- $n \leq \text{nrec}(E, E_1, E_2^*)_c - r$
- $v \sqsubseteq_{\text{val}}^{C, c_3 + \infty \cdot d} \text{nrec}(E, E_1, E_2^*)_p$

We know  $N_1[(\cdot)/x] \sqsubseteq_{\text{val}}^{C, c_3} E_1(\cdot)$ , since  $(\cdot) \sqsubseteq_{\text{val}}^{1, 0} ()$ , and so:

- $n \leq (E_1(\cdot))_c - r$
- $v \sqsubseteq_{\text{val}}^{C, c_3} (E_1(\cdot))_p$

Since  $0 \sqsubseteq_{\text{val}}^{\mathbb{N}, 0} E$ ,  $0 \leq_{\mathbb{N}} E$ , and so  $E_1(\cdot) \leq \text{nrec}(0, E_1, E_2^*) \leq \text{nrec}(E, E_1, E_2^*)$ .

- ( $n > 0$ ): Suppose that  $\text{nrec}(S(\bar{n}), \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)) \downarrow^{(n', r)} v$  by way of

$$N'_2[(\bar{n}, \lambda z.\text{nrec}(\bar{n}, \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)))] \downarrow^{(n', r)} v.$$

We must show that:

- $n' \leq \text{nrec}(E, E_1, E_2^*)_c - r$ ;
- $v \sqsubseteq_{\text{val}}^{C, c_3 + \infty \cdot d + r} \text{nrec}(E, E_1, E_2^*)_p$ .

Since  $S(\bar{n}) \sqsubseteq_{\text{val}}^{\mathbb{N}, 0} E$ , there is an  $E'$  so that  $S(E') \leq_{\mathbb{N}} E$ , and  $\bar{n} \sqsubseteq_{\text{val}}^{\mathbb{N}, 0} E'$ . For notational convenience, let  $E^* = (E', \text{nrec}(E', E_1, E_2^*))$ . Note that  $E' \leq \pi_1 E^*$ , and that  $\text{nrec}(E', E_1, E_2^*) \leq \pi_2 E^*$ . By IH,  $\text{nrec}(\bar{n}, \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)) \sqsubseteq_{\text{val}}^{C, c_3 + \infty \cdot d} \text{nrec}(E', E_1, E_2^*)$ , and thus by weakening  $\text{nrec}(\bar{n}, \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)) \sqsubseteq_{\text{val}}^{C, c_3 + \infty \cdot d} \pi_2 E^*$ . For some variable  $z$  not free in the term on the left,  $\lambda z.\text{nrec}(\bar{n}, \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)) \sqsubseteq_{\text{val}}^{1-\circ C, c_3 + \infty \cdot d} \lambda z.\pi_2 E^*$ , and so  $(\bar{n}, \lambda z.\text{nrec}(\bar{n}, \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2))) \sqsubseteq_{\text{val}}^{\mathbb{N} \otimes (1-\circ C), c_3 + \infty \cdot d} (\pi_1 E^*, \lambda z.\pi_2 E^*)$ , and since



$\lambda x.N'_2 \sqsubseteq_{\text{val}}^{\mathbb{N} \times (1-\circ C) \rightarrow \circ C, d} E_2$ , using the fact that  $\infty \cdot d + d = \infty \cdot d$

$$N'_2[(\bar{n}, \lambda z.\text{nrec}(\bar{n}, \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)))/x] \sqsubseteq^{C, c_3 + \infty \cdot d} E_2(\pi_1 E^*, \lambda z.\pi_2 E^*)$$

but,

$$\begin{aligned} E_2(\pi_1 E^*, \lambda z.\pi_2 E^*) &\leq (\lambda p.E_2(\pi_1 p, \lambda z.\pi_2 p))E^* \\ &= E_2^*(E', \text{nrec}(E', E_1, E_2^*)) \\ &\leq \text{nrec}(S(E'), E_1, E_2^*) \\ &\leq \text{nrec}(E, E_1, E_2^*) \end{aligned}$$

and so we are done by weakening.  $\square$

**THEOREM 3.7 (List(A)-RECURSOR).** *If  $\lambda x.N'_1 \sqsubseteq_{\text{val}}^{1-\circ C, c_1} E_1$  and  $\lambda x.N'_2 \sqsubseteq_{\text{val}}^{A \otimes (\text{List}(A) \& C) \rightarrow \circ C, c_2} E_2$ , then for all values  $\cdot \vdash_d v : \text{List}(A)$  such that  $v \sqsubseteq_{\text{val}}^{\text{List}(A), d} E$ , we have that  $\text{lrec}(v, \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)) \sqsubseteq^{C, c_1 + d + \infty \cdot c_2} \text{lrec}(E, E_1, \lambda x.E_2(\pi_1 x, ((0, \pi_1 \pi_2 x), \pi_2 \pi_2 x)))$*

**PROOF.** We proceed by induction on the derivation of  $\cdot \vdash_d v : \text{List}(A)$ . First, suppose  $v = []$ . To show that  $\text{lrec}([], \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)) \sqsubseteq^{C, c_1 + d + \infty \cdot c_2} \text{lrec}(E, E_1, \lambda x.E_2(\pi_1 x, ((0, \pi_1 \pi_2 x), \pi_2 \pi_2 x)))$ , assume that  $\text{lrec}([], \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)) \downarrow^{(n, r)} v$ . By inversion, it was by way of  $N'_1[()]/x \downarrow^{(n, r)} v$ . It suffices to show

- $n \leq \text{lrec}(E, E_1, \lambda x.E_2(\pi_1 x, ((0, \pi_1 \pi_2 x), \pi_2 \pi_2 x)))_c - r$
- $v \sqsubseteq_{\text{val}}^{C, c_1 + d + \infty \cdot c_2 + r} \text{lrec}(E, E_1, \lambda x.E_2(\pi_1 x, ((0, \pi_1 \pi_2 x), \pi_2 \pi_2 x)))_p$

Since  $() \leq_1 ()$ ,  $() \sqsubseteq_{\text{val}}^{1, d} ()$ , so  $N'_1[()]/x \sqsubseteq^{C, c_1 + d} E_1()$ , and so

- $n \leq (E_1())_c - r$
- $v \sqsubseteq_{\text{val}}^{C, c_1 + d + r} (E_1())_p$

But,  $\infty \cdot c_2 > 0$  since  $c_2 > 0$ , and so by credit weakening,  $v \sqsubseteq_{\text{val}}^{C, c_1 + d + \infty \cdot c_2 + r} (E_1())_p$ . Note that, by assumption,  $[] \sqsubseteq_{\text{val}}^{1, d} E$ , which means that  $[] \leq_{\text{List}(\langle\langle A \rangle\rangle)} E$ . So,

$$E_1() \leq \text{lrec}([], E_1, \lambda x.E_2(\pi_1 x, ((0, \pi_1 \pi_2 x), \pi_2 \pi_2 x))) \leq \text{lrec}(E, E_1, \lambda x.E_2(\pi_1 x, ((0, \pi_1 \pi_2 x), \pi_2 \pi_2 x)))$$

and so we are done by weakening.

Otherwise, suppose  $v = v_1 :: v_2$ . To show that  $\text{lrec}(v_1 :: v_2, \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)) \sqsubseteq^{C, c_1 + d + \infty \cdot c_2} \text{lrec}(E, E_1, \lambda x.E_2(\pi_1 x, ((0, \pi_1 \pi_2 x), \pi_2 \pi_2 x)))$ , suppose  $\text{lrec}(v_1 :: v_2, \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)) \downarrow^{(n, r)} v$ . By inversion, it was by  $N'_2[(v_1, \langle v_2, \text{lrec}(v_2, \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)) \rangle)/x] \downarrow^{(n, r)} v$ . It suffices to show:

- $n \leq \text{lrec}(E, E_1, \lambda x.E_2(\pi_1 x, ((0, \pi_1 \pi_2 x), \pi_2 \pi_2 x)))_c - r$
- $v \sqsubseteq_{\text{val}}^{C, c_1 + d + \infty \cdot c_2 + r} \text{lrec}(E, E_1, \lambda x.E_2(\pi_1 x, ((0, \pi_1 \pi_2 x), \pi_2 \pi_2 x)))_p$

Since  $v_1 :: v_2 \sqsubseteq_{\text{val}}^{\text{List}(A), d} E$ , there are  $d_1, d_2 \geq 0$  such that  $d_1 + d_2 = d$ , along with  $E', E''$  such that  $v_1 \sqsubseteq_{\text{val}}^{A, d_1} E'$  and  $v_2 \sqsubseteq_{\text{val}}^{\text{List}(A), d_2} E''$ , and  $E' :: E'' \leq E$

By IH,  $\text{lrec}(v_2, \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)) \sqsubseteq^{C, c_1 + d_2 + \infty \cdot c_2} \text{lrec}(E'', E_1, \dots)$ . Since  $v_2 \sqsubseteq_{\text{val}}^{\text{List}(A), d_2} E''$ ,  $v_2 \sqsubseteq_{\text{val}}^{\text{List}(A), d_2} (0, E'')$ , and since  $c_1 + \infty \cdot c_2 \geq 0$ , we have by credit weakening that  $v_2 \sqsubseteq_{\text{val}}^{\text{List}(A), c_1 + d_2 + \infty \cdot c_2} (0, E'')$ . So,  $\langle v_2, \text{lrec}(v_2, \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)) \rangle \sqsubseteq_{\text{val}}^{\text{List}(A) \& C, c_1 + d_2 + \infty \cdot c_2} ((0, E''), \text{lrec}(E'', E_1, \dots))$ . Further, using the fact that  $d_1 + d_2 = d$ ,

$$\begin{aligned} & (v_1, \langle v_2, \text{lrec}(v_2, \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)) \rangle) \sqsubseteq_{\text{val}}^{A \otimes (\text{List}(A) \& C), c_1 + d + \infty \cdot c_2} \\ & (E', ((0, E''), \text{lrec}(E'', E_1, \dots))) \end{aligned}$$

Thus, since  $\lambda x.N'_2 \sqsubseteq_{\text{val}}^{A \otimes (\text{List}(A) \& C) \multimap C, c_2} E_2$ , we have (using the fact that  $c_2 + \infty \cdot c_2 = \infty \cdot c_2$ )

$$N'_2[(v_1, \langle v_2, \text{lrec}(v_2, \lambda x.N'_1, \text{save}_0^\infty(\lambda x.N'_2)) \rangle)/x] \sqsubseteq^{C, c_1 + d + \infty \cdot c_2} E_2(E', ((0, E''), \text{lrec}(E'', E_1, \dots)))$$

By definition, this means that

- $n \leq (E_2(E', ((0, E''), \text{lrec}(E'', E_1, \dots))))_c - r$
- $v \sqsubseteq_{\text{val}}^{c_1 + d + \infty \cdot c_2 + r} (E_2(E', ((0, E''), \text{lrec}(E'', E_1, \dots))))_p$

We then compute:

$$\begin{aligned} & E_2(E', ((0, E''), \text{lrec}(E'', E_1, \dots))) \\ & \leq (\lambda x.E_2(\pi_1 x, ((0, \pi_1 \pi_2 x), \pi_2 \pi_2 x))) (E', (E'', \text{lrec}(E'', E_1, \lambda x.E_2(\pi_1 x, ((0, \pi_1 \pi_2 x), \pi_2 \pi_2 x)))))) \\ & \leq \text{lrec}(E' :: E'', E_1, \lambda x.E_2(\pi_1 x, ((0, \pi_1 \pi_2 x), \pi_2 \pi_2 x)))s) \\ & \leq \text{lrec}(E, E_1, \lambda x.E_2(\pi_1 x, ((0, \pi_1 \pi_2 x), \pi_2 \pi_2 x))) \end{aligned}$$

and hence we are done by weakening.  $\square$

**THEOREM 3.8 (BOUNDING THEOREM).** *If  $\Gamma \vdash_f M : A$ , then  $M \sqsubseteq^A \|M\|$*

**PROOF.** By induction on  $\Gamma \vdash_f M : A$ .

- (!-I) Let  $\Gamma \vdash_g \text{save}_l^k M : !_l^k A$ . By inversion, we have  $\Gamma \vdash_f M : A$  with  $kf + l \leq g$ . Let  $\theta \sqsubseteq_{\text{sub}}^{\Gamma, \sigma} \Theta$ . To show  $\text{save}_l^k M[\theta] \sqsubseteq_{\text{val}}^{!_l^k A, g[\sigma]} (k \|M\| [\Theta]_c, \|M\| [\Theta]_p)$ , it suffices to show  $\text{save}_l^k M[\theta] \sqsubseteq_{\text{val}}^{!_l^k A, kf[\sigma] + l} (k \|M\| [\Theta]_c, \|M\| [\Theta]_p)$  by credit weakening. So, let  $\text{save}_l^k M \downarrow^{(n, kr)} \text{save}_l^k v$  by way of  $M \downarrow^{(n, r)} v$ . It suffices to show, using the fact that  $kf[\sigma] + l + kr = k(f[\sigma] + r) + l$
- $n \leq k \|M\| [\Theta]_c - kr$
  - $\text{save}_l^k v \sqsubseteq_{\text{val}}^{!_l^k A, k(f[\sigma] + r) + l} \|M\| [\Theta]_p$
- To show  $\text{save}_l^k v \sqsubseteq_{\text{val}}^{!_l^k A, k(f[\sigma] + r) + l} \|M\| [\Theta]_p$ , it suffices to provide  $d \geq 0$  such that  $kd + l \leq k(f[\sigma] + r) + l$ , and  $v \sqsubseteq_{\text{val}}^{A, d} \|M\| [\Theta]_p$ . By IH,  $M[\theta] \sqsubseteq^{A, f[\sigma]} \|M\| [\Theta]$ , which means that
- $n \leq \|M\| [\Theta]_c - r$
  - $v \sqsubseteq_{\text{val}}^{A, f[\sigma] + r} \|M\| [\Theta]_p$
- So,  $d = f[\sigma] + r$ , and the inequality  $n \leq kn \leq k \|M\| [\Theta]_c - kr$  follows by multiplying the above one by  $k$  (since  $k \geq 1$ ).
- (!-E) Let  $\Gamma \vdash_{kf+g} \text{transfer}_{k'} !_l^k x = M \text{ to } N : C$ . By inversion, we have that  $\Gamma \vdash_f M : !_l^k A$ , as well as  $\Gamma, x : A \vdash_{g+k'(kx+l)} N : C$ . Suppose  $\theta \sqsubseteq_{\text{sub}}^{\Gamma, \sigma} \Theta$ . We need to show that  $\text{transfer}_{k'} !_l^k x = M[\theta] \text{ to } N[\theta] \sqsubseteq^{C, kf[\sigma] + g[\sigma]} \|M\| [\Theta]_c +_c \|N\| [\Theta, \|M\| [\Theta]_p/x]$ . Suppose  $\text{transfer}_{k'} !_l^k x = M[\theta] \text{ to } N[\theta] \downarrow^{(n_1 + n_2, k'r_1 + r_2)} v$ . By inversion, it was by  $M[\theta] \downarrow^{(n_1, r_1)} \text{save}_l^k v_1$  and  $N[\theta, v_1/x] \downarrow^{(n_2, r_2)} v$ . It suffices to show that
- $n_1 + n_2 \leq k' \|M\| [\Theta]_c + \|N\| [\Theta, \|M\| [\Theta]_p/x]_c - (k'r_1 + r_2)$
  - $v \sqsubseteq_{\text{val}}^{C, k'f[\sigma] + g[\sigma] + k'r_1 + r_2} \|N\| [\Theta, \|M\| [\Theta]_p/x]_p$
- By IH, we have that  $M[\theta] \sqsubseteq_{\text{val}}^{!_l^k A, f[\sigma]} \|M\| [\Theta]$ , which means that there are  $b_1, c_1$  with  $b_1 + c_1 = f[\sigma]$  and
- $n_1 \leq \|M\| [\Theta]_c - r_1$
  - $\text{save}_l^k v_1 \sqsubseteq_{\text{val}}^{!_l^k A, f[\sigma] + r_1} \|M\| [\Theta]_p$

Since  $\text{save}_l^k v_1 \sqsubseteq_{\text{val}}^{!k A, f[\sigma] + r_1} \|M\| [\Theta]_p$ , there is a  $d \geq 0$  such that  $kd + l \leq f[\sigma] + r_1$ , and  $v_1 \sqsubseteq_{\text{val}}^{A, d} \|M\| [\Theta]_p$ . Thus,  $(\theta, v_1/x) \sqsubseteq_{\text{sub}}^{(\Gamma, x:A), (\sigma, x \mapsto d)} (\Theta, \|M\| [\Theta]_p/x)$ , and so by IH,  $N[\theta, v_1/x] \sqsubseteq^{C, g[\sigma] + k'(kd+l)} \|N\| [\Theta, \|M\| [\Theta]_p/x]$ . By credit weakening, since  $kd + l \leq f[\sigma] + r_1$ ,  $N[\theta, v_1/x] \sqsubseteq^{C, g[\sigma] + k'(f[\sigma] + r_1)} \|N\| [\Theta, \|M\| [\Theta]_p/x]$ . This gives us that

$$- n_2 \leq \|N\| [\Theta, \|M\| [\Theta]_p/x]_c - r_2$$

$$- v \sqsubseteq_{\text{val}}^{C, g[\sigma] + k'(f[\sigma] + r_1) + r_2} \|N\| [\Theta, \|M\| [\Theta]_p/x]_p$$

To establish the desired inequality, we multiply the first inequality by  $k'$ , to find that  $k'n_1 \leq k' \|M\| [\Theta]_c - k'r_1$ . But  $k' \geq 1$ , so  $n_1 \leq k'n_1 \leq k' \|M\| [\Theta]_c - k'r_1$ . Therefore,  $n_1 + n_2 \leq k' \|M\| [\Theta]_c + \|N\| [\Theta, \|M\| [\Theta]_p/x]_c - (k'r_1 + r_2)$  as required. For value bounding, we note that  $g[\sigma] + k'(f[\sigma] + r_1) + r_2 = k'f[\sigma] + g[\sigma] + k'r_1 + r_2$ , and are done.

(spend) Let  $\Gamma \vdash_{f+l} \text{spend}_l M : A$ . By inversion,  $\Gamma \vdash_f M : A$ . To show  $\text{spend}_l M \sqsubseteq^A (-l) +_c \|M\|$ , suppose  $\theta \sqsubseteq_{\text{sub}}^{\Gamma, \sigma} \Theta$ . To show  $\text{spend}_l M[\theta] \sqsubseteq^{A, f[\sigma] + l} (-l) +_c \|M\| [\Theta]$ , suppose  $\text{spend}_l M[\theta] \downarrow^{(n, r-l)} v$ . By inversion we also have that  $M[\theta] \downarrow^{(n, r)} v$ . It suffices to show

$$- n \leq -l + \|M\| [\Theta]_c - (r - l)$$

$$- v \sqsubseteq_{\text{val}}^{A, f[\sigma] + l + r - l} \|M\| [\Theta]_p$$

or, canceling, it suffices to show  $n \leq \|M\| [\Theta]_c - r$  and  $v \sqsubseteq_{\text{val}}^{A, f[\sigma] + r} \|M\| [\Theta]_p$ , which is precisely what we get from the IH.

(create) Let  $\Gamma \vdash_f \text{create}_l M : A$ . By inversion,  $\Gamma \vdash_{f+l} M : A$ . To show  $\text{create}_l M \sqsubseteq^A l +_c \|M\|$ , suppose  $\theta \sqsubseteq_{\text{sub}}^{\Gamma, \sigma} \Theta$ , to show  $\text{create}_l M[\theta] \sqsubseteq^{A, f[\sigma] + l} l +_c \|M\| [\Theta]$ , suppose  $\text{create}_l M[\theta] \downarrow^{(n, r+l)} v$ . By inversion,  $M[\theta] \downarrow^{(n, r)} v$ . It suffices to show

$$- n \leq l + \|M\| [\Theta]_c - (r + l)$$

$$- v \sqsubseteq_{\text{val}}^{f[\sigma] + r + l} \|M\| [\Theta]_p$$

By IH, we have that  $M[\theta] \sqsubseteq^{A, f[\sigma] + l} \|M\| [\Theta]$ , so

$$- n \leq \|M\| [\Theta]_c - r$$

$$- v \sqsubseteq_{\text{val}}^{A, f[\sigma] + l + r} \|M\| [\Theta]_p$$

and so we are done, canceling the  $l$ s in the first inequality.

(tick) Immediate from IH, canceling 1s.

( $\multimap$ -I) Let  $\Gamma \vdash_f \lambda x. M : A \multimap B$ . By inversion,  $\Gamma, x : A \vdash_{f+x} M : B$ . Let  $\theta \sqsubseteq_{\text{sub}}^{\Gamma, \sigma} \Theta$ . To show  $\lambda x. M[\theta] \sqsubseteq^{A \multimap B, f[\sigma]} (0, \lambda x. \|M\| [\Theta])$ , let  $\lambda x. M[\theta] \downarrow^{(0, 0)} \lambda x. M[\theta]$ . The first condition is trivial ( $0 \leq 0$ ). We need to show that  $\lambda x. M \sqsubseteq_{\text{val}}^{f[\sigma]} \lambda x. \|M\| [\Theta]$ . Let  $v \sqsubseteq_{\text{val}}^{A, d} E$ . We must show that  $M[\theta, v/x] \sqsubseteq^{B, f[\sigma] + d} (\lambda x. \|M\| [\Theta])E$ , or by weakening, that  $M[\theta, v/x] \sqsubseteq^{B, f[\sigma] + d} \|M\| [\Theta, E/x]$ . But, since  $v \sqsubseteq_{\text{val}}^{A, d} E$ , we have  $(\theta, v/x) \sqsubseteq_{\text{sub}}^{(\Gamma, x:A), (\sigma, x \mapsto d)} (\Theta, E/x)$ , and so by IH,  $M[\theta, v/x] \sqsubseteq^{B, f[\sigma] + d} \|M\| [\Theta, E/x]$ , as required.

( $\multimap$ -E) Let  $\Gamma \vdash_{f+g} MN : B$ . Inversion gives  $\Gamma \vdash_f M : A \multimap B$  and  $\Gamma \vdash_g N : A$ . Let  $\theta \sqsubseteq_{\text{sub}}^{\Gamma, \sigma} \Theta$ . We must show  $M[\theta] N[\theta] \sqsubseteq^{B, f[\sigma] + g[\sigma]} (\|M\| [\Theta]_c + \|N\| [\Theta]_c) +_c \|M\| [\Theta]_p \|N\| [\Theta]_p$ . Suppose  $M[\theta] N[\theta] \downarrow^{(n_1 + n_2 + n_3, r_1 + r_2 + r_3)} v$ . Inversion gives us that  $M[\theta] \downarrow^{(n_1, r_1)} \lambda x. M'$ ,  $N[\theta] \downarrow^{(n_2, r_2)} v_1$ , and  $M'[v_1/x] \downarrow^{(n_3, r_3)} v$ . It remains to show that

$$- n_1 + n_2 + n_3 \leq \|M\| [\Theta]_c + \|N\| [\Theta]_c + (\|M\| [\Theta]_p \|N\| [\Theta]_p)_c - (r_1 + r_2 + r_3)$$

$$- v \sqsubseteq_{\text{val}}^{B, f[\sigma] + g[\sigma] + r_1 + r_2 + r_3} (\|M\| [\Theta]_p \|N\| [\Theta]_p)_p$$

By the IH applied to  $\Gamma \vdash_f M : A \multimap B$ , we know that  $M[\theta] \sqsubseteq^{A \multimap B, f[\sigma]} \|M\| [\Theta]$ , so

$$- n_1 \leq \|M\| [\Theta]_c - r_1$$

$$- \lambda x. M' \sqsubseteq_{\text{val}}^{A \multimap B, f[\sigma] + r_1} \|M\| [\Theta]_p$$

Again applying the IH to  $\Gamma \vdash_g N : A$ , we know  $N[\theta] \sqsubseteq^{A, g[\sigma]} \|N\| [\Theta]$ , so

$$- n_2 \leq \|N\| [\Theta]_c - r_2$$

- $v_1 \sqsubseteq_{\text{val}}^{A, g[\sigma] + r_2} \|N\| [\Theta]_p$
- But since  $\lambda x. M' \sqsubseteq_{\text{val}}^{A \multimap B, f[\sigma] + r_1} \|M\| [\Theta]_p$  and  $v_1 \sqsubseteq_{\text{val}}^{A, g[\sigma] + r_2} \|N\| [\Theta]_p$ , we have  $M'[v_1/x] \sqsubseteq_{\text{val}}^{B, f[\sigma] + g[\sigma] + r_1 + r} \|M\| [\Theta]_p \|N\| [\Theta]_p$ , which means that:
  - $n_3 \leq (\|M\| [\Theta]_p \|N\| [\Theta]_p)_c - r_3$
  - $v \sqsubseteq_{\text{val}}^{B, f[\sigma] + g[\sigma] + r_1 + r_2 + r_3} (\|M\| [\Theta]_p \|N\| [\Theta]_p)_p$

We add the inequalities together, and are done.

- ( $\otimes$ -I) Let  $\Gamma \vdash_{f+g_1} (M_1, M_2) : A_1 \otimes A_2$ . By inversion, we have that  $\Gamma \vdash_{f_i} M_i : A_i$  for  $i = 1, 2$ . Let  $\theta \sqsubseteq_{\text{sub}}^{\Gamma, \sigma} \Theta$ . Towards proving  $(M_1[\theta], M_2[\theta]) \sqsubseteq_{\text{val}}^{A_1 \otimes A_2, f_1[\sigma] + f_2[\sigma]} (\|M_1\| [\Theta]_c + \|M_2\| [\Theta]_c, (\|M_1\| [\Theta]_p, \|M_2\| [\Theta]_p))$ , assume  $(M_1[\theta], M_2[\theta]) \downarrow^{(n_1+n_2, r_1+r_2)} (v_1, v_2)$ . By inversion, it must also be that  $M_i[\theta] \downarrow^{(n_i, r_i)} v_i$  for  $i = 1, 2$ . It suffices to show:

- $n_1 + n_2 \leq \|M_1\| [\Theta]_c + \|M_2\| [\Theta]_c - (r_1 + r_2)$
- $(v_1, v_2) \sqsubseteq_{\text{val}}^{A_1 \otimes A_2, f_1[\sigma] + f_2[\sigma] + r_1 + r_2} (\|M_1\| [\Theta]_p, \|M_2\| [\Theta]_p)$

By IH, we have that, for  $i \in \{1, 2\}$

- $n_i \leq \|M_i\| [\Theta]_c - r_i$
- $v_i \sqsubseteq_{\text{val}}^{A_i, f[\sigma] + r_i} \|M_i\| [\Theta]_p$

Adding the two inequalities and applying the definition of value bounding at  $\otimes$ , we are done.

- ( $\otimes$ -E) Let  $\Gamma \vdash_{k'f+g} \text{split}_{k'}(M, x.y.N) : C$ . Inversion gives  $\Gamma \vdash_f M : A \otimes B$ , and  $\Gamma, x : A, y : B \vdash_{g+k'(x+y)} N : C$ . Let  $\theta \sqsubseteq_{\text{sub}}^{\Gamma, \sigma} \Theta$ . We must show that

$$\text{split}_{k'}(M[\theta], x.y.N[\theta]) \sqsubseteq_{\text{val}}^{C, k'f[\sigma] + g[\sigma]} k' \|M\| [\Theta]_c + \|N\| [\Theta, \pi_1 \|M\| [\Theta]_p/x, \pi_2 \|M\| [\Theta]_p/y]$$

Suppose that  $\text{split}_{k'}(M[\theta], x.y.N[\theta]) \downarrow^{(n_1+n_2, k'r_1+r_2)} v$  by way of  $M[\theta] \downarrow^{(n_1, r_1)} (v_1, v_2)$  and  $N[\theta, v_1/x, v_2/y] \downarrow^{(n_2, r_2)} v$ . It remains to show that

- $n_1 + n_2 \leq k' \|M\| [\Theta]_c + \|N\| [\Theta, \pi_1 \|M\| [\Theta]_p/x, \pi_2 \|M\| [\Theta]_p/y]_c - (k'r_1 + r_2)$
- $v \sqsubseteq_{\text{val}}^{C, k'f[\sigma] + g[\sigma] + k'r_1 + r_2} \|N\| [\Theta, \pi_1 \|M\| [\Theta]_p/x, \pi_2 \|M\| [\Theta]_p/y]_p$

By IH,  $M[\theta] \sqsubseteq_{\text{val}}^{A \otimes B, f[\sigma]} \|M\| [\Theta]$ , so

- $n_1 \leq \|M\| [\Theta]_c - r_1$
- $(v_1, v_2) \sqsubseteq_{\text{val}}^{A \otimes B, f[\sigma] + r_1} \|M\| [\Theta]_p$

and so there are  $c_1, c_2 \geq 0$  so that  $c_1 + c_2 = f[\sigma] + r_1$  and  $v_1 \sqsubseteq_{\text{val}}^{A, c_1} \pi_1 \|M\| [\Theta]_p$  and  $v_2 \sqsubseteq_{\text{val}}^{B, c_2} \pi_2 \|M\| [\Theta]_p$ . So,  $(\theta, v_1/x, v_2/y) \sqsubseteq_{\text{sub}}^{(\Gamma, x:A, y:B), (\sigma, x \mapsto c_1, y \mapsto c_2)} (\Theta, \pi_1 \|M\| [\Theta]_p/x, \pi_2 \|M\| [\Theta]_p/y)$ .

So, by IH,  $N[\theta, v_1/x, v_2/y] \sqsubseteq_{\text{val}}^{C, g[\sigma] + k'(f[\sigma] + r_1)} \|N\| [\Theta, \pi_1 \|M\| [\Theta]_p/x, \pi_2 \|M\| [\Theta]_p/y]$ , so

- $n_2 \leq \|N\| [\Theta, \pi_1 \|M\| [\Theta]_p/x, \pi_2 \|M\| [\Theta]_p/y]_c - r_2$
- $v \sqsubseteq_{\text{val}}^{C, k'f[\sigma] + g[\sigma] + k'r_1 + r_2} \|N\| [\Theta, \pi_1 \|M\| [\Theta]_p/x, \pi_2 \|M\| [\Theta]_p/y]_p$

Then,

$$\begin{aligned} n_1 + n_2 &\leq k'n_1 + n_2 \\ &\leq k' \|M\| [\Theta]_c + \|N\| [\Theta, \pi_1 \|M\| [\Theta]_p/x, \pi_2 \|M\| [\Theta]_p/y]_c - (k'r_1 + r_2) \end{aligned}$$

as required.

- ( $\oplus$ -E) Let  $\Gamma \vdash_{k'f+g_1+g_2} \text{case}_{k'}(M, x.N_1, y.N_2) : C$ . By inversion,  $\Gamma \vdash_f M : A \oplus B$ ,  $\Gamma, x : A \vdash_{g_1+k'x} N_1 : C$ , and  $\Gamma, y : B \vdash_{g_2+k'y} N_2 : C$ . Let  $\theta \sqsubseteq_{\text{sub}}^{\Gamma, \sigma} \Theta$ . We must show that

$$\text{case}_{k'}(M[\theta], x.N_1[\theta], y.N_2[\theta]) \sqsubseteq_{\text{val}}^{C, k'f[\sigma] + g_1[\sigma] + g_2[\sigma]} k' \|M\| [\Theta]_c + \text{case}(\|M\| [\Theta]_p, x. \|N_1\|, y. \|N_2\|)$$

Because the two cases are symmetric, we consider only the following evaluation:

$\text{case}_{k'}(M[\theta], x.N_1[\theta], y.N_2[\theta]) \downarrow^{(n_1+n_2, k'r_1+r_2)} v$  by way of  $M[\theta] \downarrow^{(n_1, r_1)} \text{inl } v_1$  and  $N_1[\theta, v_1/x] \downarrow^{(n_2, r_2)} v$ . We must show that

- $n_1 + n_2 \leq k' \|M\| [\Theta]_c + \text{case}(\|M\| [\Theta]_p, x. \|N_1\|, y. \|N_2\|)_c - (k'r_1 + r_2)$

$$- v \sqsubseteq_{\text{val}}^{C, k'f[\sigma] + g_1[\sigma] + g_2[\sigma] + k'r_1 + r_2} \text{case} (\|M\| [\Theta]_p, x. \|N_1\|, y. \|N_2\|)_p$$

By IH,  $M[\theta] \sqsubseteq^{A \oplus B, f[\sigma]} \|M\| [\Theta]$ , so

$$- n_1 \leq \|M\| [\Theta]_c - r_1$$

$$- \text{inl } v_1 \sqsubseteq_{\text{val}}^{A \oplus B, f[\sigma] + r_1} \|M\| [\Theta]_p$$

so there is an  $E$  such that  $\text{inl } E \leq_{\langle A \rangle + \langle B \rangle} \|M\| [\Theta]_p$  and  $v_1 \sqsubseteq_{\text{val}}^{A, f[\sigma] + r_1} E$ . So,

$$(\theta, v_1/x) \sqsubseteq_{\text{sub}}^{(\Gamma, x:A), (\sigma, x \mapsto f[\sigma] + r_1)} (\Theta, E/x) \text{ and hence by IH, } N_1[\theta, v_1/x] \sqsubseteq_{g_1[\sigma] + k'(f[\sigma] + r_1)} \|N_1\| [\Theta, E/x], \text{ so}$$

$$- n_2 \leq \|N_1\| [\Theta, E/x]_c - r_2$$

$$- v \sqsubseteq_{\text{val}}^{C, k'f[\sigma] + g_1[\sigma] + k'r_1 + r_2} \|N_1\| [\Theta, E/x]_p$$

Since  $g_2[\sigma] \geq 0$ , we have by credit weakening that  $v \sqsubseteq_{\text{val}}^{C, k'f[\sigma] + g_1[\sigma] + g_2[\sigma] + k'r_1 + r_2} \|N_1\| [\Theta, E/x]_p$ . Then, we compute:

$$\begin{aligned} \|N_1\| [\Theta, E/x] &\leq_{\|C\|} \text{case} (\text{inl } E, x. \|N_1\| [\Theta], y. \|N_2\| [\Theta]) \\ &\leq \text{case} (\|M\| [\Theta]_p, x. \|N_1\| [\Theta], y. \|N_2\| [\Theta]) \end{aligned}$$

which gives us the value bounding condition, and again compute:

$$\begin{aligned} n_1 + n_2 &\leq k'n_1 + n_2 \\ &\leq k' \|M\| [\Theta]_c + \text{case} (\|M\| [\Theta]_p, x. \|N_1\|, y. \|N_2\|)_c - (k'r_1 + r_2) \end{aligned}$$

which gives us the cost bounding condition.

( $\oplus$ -I) The cases for  $\text{inl } M$  and  $\text{inr } M$  are symmetric, so we let  $\Gamma \vdash_f \text{inl } M : A \oplus B$ . Inversion gives  $\Gamma \vdash_f M : A$ . Let  $\theta \sqsubseteq_{\text{sub}}^{\Gamma, \sigma} \Theta$ . To show that  $\text{inl } M[\theta] \sqsubseteq^{A \oplus B, f[\sigma]} (\|M\| [\Theta]_c, \text{inl } \|M\| [\Theta]_p)$ , we let  $\text{inl } M[\theta] \downarrow^{(n, r)} \text{inl } v$ . Inverting, we have  $M[\theta] \downarrow^{(n, r)} v$ . It suffices to show that  $n \leq \|M\| [\Theta]_c - r$ , and that  $\text{inl } v \sqsubseteq_{\text{val}}^{A \oplus B, f[\sigma] + r} \text{inl } \|M\| [\Theta]_p$ . By IH we have  $n \leq \|M\| [\Theta]_c - r$ , and  $v \sqsubseteq_{\text{val}}^{A, f[\sigma] + r} \|M\| [\Theta]_p$ . So we are done by the definition of value bounding at  $\oplus$  for  $\text{inl}$ .

List (A)-I, cons) Let  $\Gamma \vdash_{f+g} M :: N : \text{List}(A)$ . By inversion,  $\Gamma \vdash_f M : A$  and  $\Gamma \vdash_g N : \text{List}(A)$ . Let  $\theta \sqsubseteq_{\text{sub}}^{\Gamma, \sigma} \Theta$ . To show that  $M :: N \sqsubseteq^{\text{List}(A), f[\sigma] + g[\sigma]} (\|M\| [\Theta]_c + \|N\| [\Theta]_c, \|M\| [\Theta]_p :: \|N\| [\Theta]_p)$ , let  $M :: N \downarrow^{(n_1 + n_2, r_1 + r_2)} v_1 :: v_2$ . By inversion,  $M \downarrow^{(n_1, r_1)} v_1$  and  $N \downarrow^{(n_2, r_2)} v_2$ . It suffices to provide  $b, c$  where  $c \geq 0$  and  $b + c = f[\sigma] + g[\sigma]$  and that

$$- n_1 + n_2 \leq \|M\| [\Theta]_c + \|N\| [\Theta]_c - (r_1 + r_2)$$

$$- v_1 :: v_2 \sqsubseteq_{\text{val}}^{\text{List}(A), f[\sigma] + g[\sigma] + r_1 + r_2} \|M\| [\Theta]_p :: \|N\| [\Theta]_p.$$

By IH,

$$- n_1 \leq \|M\| [\Theta]_c - r_1$$

$$- v_1 \sqsubseteq_{\text{val}}^{A, f[\sigma] + r_1} \|M\| [\Theta]_p$$

and by IH on  $N$ , there are  $b_2, c_2$  with  $c_2 \geq 0$  and  $b_2 + c_2 = g[\sigma]$  such that

$$- n_2 \leq \|N\| [\Theta]_c - r_2$$

$$- v_2 \sqsubseteq_{\text{val}}^{\text{List}(A), g[\sigma] + r_2} \|N\| [\Theta]_p$$

Thus, the desired inequality follows from adding the two inductively computed ones, and the value bounding relation for  $v_1 :: v_2$  is immediate by the definition.

(List (A)-E) Suppose  $\Gamma \vdash_{f+g_1+g_2} \text{lrec}(M, N_1, N_2) : C$ . By inversion,  $\Gamma \vdash_f M : \text{List}(A)$ ,  $\Gamma \vdash_{g_1} N_1 : 1 \multimap C$ ,  $\Gamma \vdash_{g_2} N_2 : !_0^\infty(A \otimes (\text{List}(A) \& C) \multimap C)$ . Let  $\theta \sqsubseteq_{\text{sub}}^{\Gamma, \sigma} \Theta$ . To show that

$$\text{lrec}(M[\theta], N_1[\theta], N_2[\theta]) \sqsubseteq^{C, f[\sigma] + g_1[\sigma] + g_2[\sigma]} (\|M\| [\Theta]_c + \|N_1\| [\Theta]_c + \|N_2\| [\Theta]_c)_c$$

$$\text{lrec}(\|M\| [\Theta]_p, \|N_1\| [\Theta]_p, \lambda(a, (as, r)). \|N_2\| [\Theta]_p(a, ((0, as), r)))$$

we break into the two evaluation cases. Firstly, suppose that  $\text{1rec}(M[\theta], N_1[\theta], N_2[\theta]) \downarrow^{(n_1+n_2+n_3+n_4, r_1+r_2+r_3+r_4)} v$  by way of  $M[\theta] \downarrow^{(n_1, r_1)} []$ ,  $N_1[\theta] \downarrow^{(n_2, r_2)} \lambda x. N'_1$ ,  $N_2[\theta] \downarrow^{(n_3, r_3)} \text{save}_0^\infty(\lambda x. N'_2)$ , and  $N'_1[()/x] \downarrow^{(n_4, r_4)} v$ . From here, denote we denote  $\lambda x. \|N_2\| [\Theta]_p(\pi_1 x, ((0, \pi_1 \pi_2 x), \pi_2 \pi_2 x))$  as  $\|N_2\|^*$ .

It suffices to show that:

$$\begin{aligned} & - n_1 + n_2 + n_3 + n_4 \leq \|M\| [\Theta]_c + \|N_1\| [\Theta]_c + \|N_2\| [\Theta]_c + \text{1rec} \left( \|M\| [\Theta]_p, \|N_1\| [\Theta]_p, \|N_2\|^* \right)_c - \\ & \quad (r_1 + r_2 + r_3 + r_4) \\ & - v \sqsubseteq_{\text{val}}^{C, f[\sigma] + g_1[\sigma] + g_2[\sigma] + r_1 + r_2 + r_3 + r_4} \text{1rec} \left( \|M\| [\Theta]_p, \|N_1\| [\Theta]_p, \|N_2\|^* \right)_p \end{aligned}$$

By IH,  $M[\theta] \sqsubseteq^{\text{List}(A), f[\sigma]} \|M\| [\Theta]$ , so

$$\begin{aligned} & - n_1 \leq \|M\| [\Theta]_c - r_1 \\ & - [] \sqsubseteq_{\text{val}}^{f[\sigma] + r_1} \|M\| [\Theta]_p \end{aligned}$$

The second condition tells us that  $[] \leq \|M\| [\Theta]_p$ . Again by IH,  $N_2[\theta] \sqsubseteq_0^{\text{1}\infty(A \otimes (\text{List}(A) \& C) \multimap C), g_2[\sigma]} \|N_2\| [\Theta]$ , so

$$\begin{aligned} & - n_3 \leq \|N_2\| [\Theta]_p - r_3 \\ & - \text{save}_0^\infty(\lambda x. N'_2) \sqsubseteq_{\text{val}}^{\text{1}\infty(A \otimes (\text{List}(A) \& C) \multimap C), g_2[\sigma] + r_3} \|N_2\| [\Theta]_p \end{aligned}$$

In particular, by preservation,  $g_2[\sigma] + r_3 \geq 0$ . Thirdly by IH,  $N_1[\theta] \sqsubseteq^{1 \multimap C, g_1[\sigma]} \|N_1\| [\Theta]$ , which means

$$\begin{aligned} & - n_2 \leq \|N_1\| [\Theta]_c - r_2 \\ & - \lambda x. N'_1 \sqsubseteq_{\text{val}}^{1 \multimap C, g_1[\sigma] + r_2} \|N_1\| [\Theta]_p \end{aligned}$$

Since  $() \leq ()$ ,  $() \sqsubseteq_{\text{val}}^{1, f[\sigma] + r_1} ()$ . Hence,  $N'_2[()/x] \sqsubseteq^{C, g_1[\sigma] + f[\sigma] + r_1 + r_2} \|N_1\| [\Theta]_p ()$ . This means that

$$\begin{aligned} & - n_4 \leq (\|N_1\| [\Theta]_p ())_c - r_4 \\ & - v \sqsubseteq_{\text{val}}^{C, f[\sigma] + g_1[\sigma] + r_1 + r_2 + r_4} (\|N_1\| [\Theta]_p ())_p \end{aligned}$$

But by credit weakening, since  $g_2[\sigma] + r_3 \geq 0$ , we have  $v \sqsubseteq_{\text{val}}^{C, f[\sigma] + g_1[\sigma] + g_2[\sigma] + r_1 + r_2 + r_3 + r_4} (\|N_1\| [\Theta]_p ())_p$ . But, we can compute:

$$\|N_1\| [\Theta]_p () \leq \text{1rec} ( [], \|N_1\| [\Theta]_p, \|N_2\| [\Theta]_p ) \leq \text{1rec} ( \|M\| [\Theta]_p, \|N_1\| [\Theta]_p, \|N_2\|^* )$$

and we are done by weakening.

Otherwise, assume  $M[\theta] \downarrow^{(n_1, r_1)} v_1 :: v_2$ ,  $N_2[\theta] \downarrow^{(n_2, r_2)} \text{save}_0^\infty(\lambda x. N'_2)$ ,  $N_1[\theta] \downarrow^{(n_3, r_3)} \lambda x. N'_1$ , and

$$N'_2[(v_1, \langle v_2, \text{1rec}(v_2, \lambda x. N'_1, \text{save}_0^\infty(\lambda x. N'_2)) \rangle)] \downarrow^{(n_4, r_4)} v$$

. Just like the previous case, it suffices to show

$$\begin{aligned} & - n_1 + n_2 + n_3 + n_4 \leq \|M\| [\Theta]_c + \|N_1\| [\Theta]_c + \|N_2\| [\Theta]_c + \text{1rec} \left( \|M\| [\Theta]_p, \|N_1\| [\Theta]_p, \|N_2\|^* \right)_c - \\ & \quad (r_1 + r_2 + r_3 + r_4) \\ & - v \sqsubseteq_{\text{val}}^{C, f[\sigma] + g_1[\sigma] + g_2[\sigma] + r_1 + r_2 + r_3 + r_4} \text{1rec} \left( \|M\| [\Theta]_p, \|N_1\| [\Theta]_p, \|N_2\| [\Theta]_p \right)_p \end{aligned}$$

By IH,  $M[\theta] \sqsubseteq^{\text{List}(A), f[\sigma]} \|M\| [\Theta]$ , so

$$\begin{aligned} & - n_1 \leq \|M\| [\Theta]_c - r_1 \\ & - v_1 :: v_2 \sqsubseteq_{\text{val}}^{\text{List}(A), f[\sigma] + r_1} \|M\| [\Theta]_p \end{aligned}$$

By the second condition, we know that there are  $d_1, d_2 \geq 0$  with  $d_1 + d_2 = f[\sigma] + r_1$ , and  $E_1, E_2$  with  $E_1 :: E_2 \leq \|M\| [\Theta]_p$  such that  $v_1 \sqsubseteq_{\text{val}}^{A, d_1} E_1$ , and  $v_2 \sqsubseteq_{\text{val}}^{\text{List}(A), d_2} E_2$ . By IH,  $N_1[\theta] \sqsubseteq^{1 \multimap C, g_1[\sigma]} \|N_1\| [\Theta]$ , so

$$\begin{aligned} & - n_3 \leq \|N_1\| [\Theta]_c - r_3 \\ & - \lambda x. N'_1 \sqsubseteq_{\text{val}}^{1 \multimap C, g_1[\sigma] + r_3} \|N_1\| [\Theta]_p \end{aligned}$$



Again by IH,  $N_2[\theta] \sqsubseteq_0^{f[\sigma](A \otimes (\text{List}(A) \& C) \multimap C), g_2[\sigma]} \|N_2\| [\Theta]$ , which means that

$$\begin{aligned} - n_2 &\leq \|N_2\| [\theta]_c - r_2 \\ - \text{save}_0^\infty (\lambda x. N'_2) &\sqsubseteq_{\text{val}}^{f[\sigma](A \otimes (\text{List}(A) \& C) \multimap C), g_2[\sigma] + r_2} \|N_2\| [\Theta]_p \end{aligned}$$

The second condition means by definition that there is a  $c \geq 0$  such that  $\infty \cdot c \leq g_2[\sigma] + r_2$ , and  $\lambda x. N'_2 \sqsubseteq_{\text{val}}^{A \otimes (\text{List}(A) \& C) \multimap C, c} \|N_2\| [\Theta]_p$ . We claim that

$$\begin{aligned} N'_2[(v_1, \langle v_2, \text{lrec}(v_2, \lambda x. N'_2, \text{save}_0^\infty (\lambda x. N'_2)) \rangle)] &\sqsubseteq^{f[\sigma] + g_1[\sigma] + g_2[\sigma] + r_1 + r_2 + r_3} \\ \|N_2\| (E_1, ((0, E_2), \text{lrec}(E_2, \|N_1\| [\Theta]_p, \|N_2\|^*))) \end{aligned}$$

To prove this claim, we split into cases on the finitude of  $g_2[\sigma] + r_2$ .

- Suppose  $g_2[\sigma] + r_2$  is finite. Then  $c = 0$ , and  $g_2[\sigma] + r_2 = 0$ , and so by the list recursor lemma with  $E_1 = \|N_1\| [\Theta]_p$ ,  $E_2 = \|N_2\| [\Theta]_p$ , and  $E = E_2$ , we have that  $\text{lrec}(v_2, \lambda x. N'_2, \text{save}_0^\infty (\lambda x. N'_2)) \sqsubseteq^{C, d_2 + g_1[\sigma] + r_3} \text{lrec}(E_2, \|N_1\| [\Theta]_p, \|N_2\|^*)$ . Since  $v_2 \sqsubseteq_{\text{val}}^{\text{List}(A), d_2} E_2$ , we have that  $v_2 \sqsubseteq^{\text{List}(A), d_2} (0, E_2)$ , and by credit weakening, since  $g_1[\sigma] + r_3 \geq 0$ ,  $v_2 \sqsubseteq^{\text{List}(A), d_2 + g_1[\sigma] + r_3} (0, E_2)$ . Thus:

$$\begin{aligned} \langle v_2, \text{lrec}(v_2, \lambda x. N'_2, \text{save}_0^\infty (\lambda x. N'_2)) \rangle &\sqsubseteq^{\text{List}(A) \& C, d_2 + g_1[\sigma] + r_3} \\ ((0, E_2), \text{lrec}(E_2, \|N_1\| [\Theta]_p, \|N_2\|^*)) \end{aligned}$$

Next, since  $v_1 \sqsubseteq_{\text{val}}^{A, d_1} E_1$ ,  $d_1 + d_2 = f[\sigma] + r_1$ , and  $\lambda x. N'_2 \sqsubseteq_{\text{val}}^{A \otimes (\text{List}(A) \& C), 0} \|N_2\| [\Theta]_p$ ,

$$\begin{aligned} N'_2[(v_1, \langle v_2, \text{lrec}(v_2, \lambda x. N'_2, \text{save}_0^\infty (\lambda x. N'_2)) \rangle)/x] &\sqsubseteq^{f[\sigma] + g_1[\sigma] + r_1 + r_3} \\ \|N_2\| [\Theta]_p (E_1, ((0, E_2), \text{lrec}(E_2, \|N_1\| [\Theta]_p, \|N_2\|^*))) \end{aligned}$$

Which is exactly what we wanted to show, since  $g_2[\sigma] + r_2 = 0$ .

- Suppose  $g_2[\sigma] + r_2 = \infty$ . Then, there is a  $c \geq 0$  such that  $\infty \cdot c \leq \infty$ , and  $\lambda x. N'_2 \sqsubseteq_{\text{val}}^{A \otimes (\text{List}(A) \& C) \multimap C, c} \|N_2\| [\Theta]_p$ . By credit weakening, we may assume  $c > 0$ . By the list recursor lemma,  $\text{lrec}(v_2, \lambda x. N'_2, \text{save}_0^\infty (\lambda x. N'_2)) \sqsubseteq^{C, d_2 + g_1[\sigma] + r_3 + \infty \cdot c} \text{lrec}(E_2, \|N_1\| [\Theta]_p, \|N_2\|^*)$ . By the same reasoning as in the previous case,

$$\begin{aligned} (v_1, \langle v_2, \text{lrec}(v_2, \lambda x. N'_2, \text{save}_0^\infty (\lambda x. N'_2)) \rangle) &\sqsubseteq^{A \otimes (\text{List}(A) \& C), f[\sigma] + g_1[\sigma] + r_1 + r_3} \\ (E_1, ((0, E_2), \text{lrec}(E_2, \|N_1\| [\Theta]_p, \|N_2\|^*))) \end{aligned}$$

Then, since  $\infty \cdot c + c = \infty \cdot c$ ,

$$\begin{aligned} N'_2[(v_1, \langle v_2, \text{lrec}(v_2, \lambda x. N'_2, \text{save}_0^\infty (\lambda x. N'_2)) \rangle)/x] &\sqsubseteq^{C, f[\sigma] + g_1[\sigma] + r_1 + r_3 + \infty \cdot c} \\ \|N_2\| [\Theta]_p (E_1, ((0, E_2), \text{lrec}(E_2, \|N_1\| [\Theta]_p, \|N_2\|^*))) \end{aligned}$$

which,  $\infty \cdot c \leq g_2[\sigma] + r_2$ , gives us our goal by credit weakening.

From this result, we have by definition that

$$\begin{aligned} - n_4 &\leq (\|N_2\| [\Theta]_p (E_1, ((0, E_2), \text{lrec}(E_2, \|N_1\| [\Theta]_p, \|N_2\|^*))))_c - r_4 \\ - v &\sqsubseteq_{\text{val}}^{C, f[\sigma] + g_1[\sigma] + g_2[\sigma] + r_1 + r_2 + r_3 + r_4} (\|N_2\| [\Theta]_p (E_1, ((0, E_2), \text{lrec}(E_2, \|N_1\| [\Theta]_p, \|N_2\|^*))))_p \end{aligned}$$

Then we can compute:

$$\begin{aligned} \|N_2\| [\Theta]_p (E_1, ((0, E_2), \text{lrec}(E_2, \|N_1\| [\Theta]_p, \|N_2\|^*))) &\leq \|N_2\|^* (E_1, (E_2, \text{lrec}(E_2, \|N_1\| [\Theta]_p, \|N_2\|^*))) \\ &\leq \text{lrec}(E_1 :: E_2, \|N_1\| [\Theta]_p, \|N_2\|^*) \\ &\leq \text{lrec}(\|M\| [\Theta]_p, \|N_1\| [\Theta]_p, \|N_2\|^*) \end{aligned}$$

and so we are done by weakening.

(N-E) Suppose  $\Gamma \vdash_{f+g_1+g_2} \text{nrec}(M, N_1, N_2) : C$ . By inversion, we have that  $\Gamma \vdash_f M : \mathbb{N}$ ,  $\Gamma \vdash_{g_1} N_1 : 1 \multimap C$ , and  $\Gamma \vdash_{g_2} N_2 : !_0^\infty(\mathbb{N} \otimes (1 \multimap C) \multimap C)$ . Let  $\theta \sqsubseteq_{\text{sub}}^{\Gamma, \sigma} \Theta$ . For convenience, let  $\|N_2\| [\Theta]_p^* = \lambda p. \|N_2\| [\Theta]_p(\pi_1 p, \lambda z. \pi_2 p)$ . We must show:

$$\begin{aligned} & \text{nrec}(M[\theta], N_1[\theta], N_2[\theta]) \sqsubseteq^{C, f[\sigma]+g_1[\sigma]+g_2[\sigma]} \\ & (\|M\| [\Theta]_c + \|N_1\| [\Theta]_c + \|N_2\| [\Theta]_c) +_c \text{nrec}(\|M\| [\Theta]_p, \|N_1\| [\Theta]_p, \|N_2\| [\Theta]_p^*) \end{aligned}$$

In order to show this, we have two evaluation cases to consider. Suppose

$$\text{nrec}(M[\theta], N_1[\theta], N_2[\theta]) \downarrow^{(n_1+n_2+n_3+n_4, r_1+r_2+r_3+r_4)} v$$

by way of  $M[\theta] \downarrow^{(n_1, r_1)} 0$ ,  $N_1[\theta] \downarrow^{(n_2, r_2)} \lambda x. N'_1$ ,  $N_2[\theta] \downarrow^{(n_3, r_3)} v'$ , and  $N'_1[()]/x \downarrow^{(n_4, r_4)} v$ . It suffices to show that:

$$\begin{aligned} - & n_1 + n_2 + n_3 + n_4 \leq b + \|M\| [\Theta]_c + \|N_1\| [\Theta]_c + \|N_2\| [\Theta]_c + \\ & \text{nrec}(\|M\| [\Theta]_p, \|N_1\| [\Theta]_p, \|N_2\| [\Theta]_p^*)_c - (r_1 + r_2 + r_3 + r_4) \\ - & v \sqsubseteq_{\text{val}}^{C, f[\sigma]+g_1[\sigma]+g_2[\sigma]+r_1+r_2+r_3+r_4} \text{nrec}(\|M\| [\Theta]_p, \|N_1\| [\Theta]_p, \|N_2\| [\Theta]_p^*)_p \end{aligned}$$

By IH,  $M[\theta] \sqsubseteq^{\mathbb{N}, f[\sigma]} \|M\| [\Theta]$ , so

$$- n_1 \leq \|M\| [\Theta]_c - r_1$$

$$- 0 \sqsubseteq_{\text{val}}^{\mathbb{N}, f[\sigma]+r_1} \|M\| [\Theta]_p$$

since  $0 \sqsubseteq_{\text{val}}^{\mathbb{N}, f[\sigma]+r_1} \|M\| [\Theta]_p$ ,  $0 \leq_{\mathbb{N}} \|M\| [\Theta]_p$ . By IH,  $N_1[\theta] \sqsubseteq^{1 \multimap C, g_1[\sigma]} \|N_1\| [\Theta]$ , so

$$- n_2 \leq \|N_1\| [\Theta]_c - r_2$$

$$- \lambda x. N'_1 \sqsubseteq_{\text{val}}^{1 \multimap C, g_1[\sigma]+r_2} \|N_1\| [\Theta]_p$$

By IH,  $N_2[\theta] \sqsubseteq^{!_0^\infty(\dots), g_2[\sigma]} \|N_2\| [\Theta]$ , and so

$$- n_3 \leq \|N_2\| [\Theta]_c - r_3$$

We omit the value bounding condition since it does not factor into the rest of the proof. Since

$$() \leq_1 (), () \sqsubseteq_{\text{val}}^{1, f[\sigma]+r_1} (). \text{ So: } N'_1[()]/x \sqsubseteq^{C, f[\sigma]+g_1[\sigma]+r_1+r_2} \|N_1\| [\Theta] (). \text{ Thus,}$$

$$- n_4 \leq (\|N_1\| [\Theta] ())_c - r_4$$

$$- v \sqsubseteq_{\text{val}}^{C, f[\sigma]+g_1[\sigma]+r_1+r_2+r_4} (\|N_1\| [\Theta] ())_p$$

Since  $g_2[\sigma] + r_3 \geq 0$ , we know by credit weakening,  $v \sqsubseteq_{\text{val}}^{C, f[\sigma]+g_1[\sigma]+g_2[\sigma]+r_1+r_2+r_3+r_4}$

$(\|N_1\| [\Theta] ())_p$ . Since  $0 \leq \|M\| [\Theta]_p$ , we compute:

$$\begin{aligned} & \|N_1\| [\Theta] () \leq_C \text{nrec}(0, \|N_1\| [\Theta]_p, \|N_2\| [\Theta]_p^*) \\ & \leq \text{nrec}(\|M\| [\Theta]_p, \|N_1\| [\Theta]_p, \|N_2\| [\Theta]_p^*)_p \end{aligned}$$

So we are done by weakening.

Suppose  $\text{nrec}(M[\theta], N_1[\theta], N_2[\theta]) \downarrow^{(n_1+n_2+n_3+n_4, r_1+r_2+r_3+r_4)} v$  by way of  $M[\theta] \downarrow^{(n_1, r_1)} S(v_1)$ ,  $N_2[\theta] \downarrow^{(n_2, r_2)} \text{save}_0^\infty \lambda x. N'_2$ ,  $N_1[\theta] \downarrow^{(n_3, r_3)} \lambda x. N'_1$ , and

$$N'_2[(v_1, \text{nrec}(v_1, \lambda z. (\text{nrec}(v_1, \lambda x. N'_1, \text{save}_0^\infty \lambda x. N'_2)), )) / x] \downarrow^{(n_4, r_4)} v$$

$$\begin{aligned} - & n_1 + n_2 + n_3 + n_4 \leq b + \|M\| [\Theta]_c + \|N_1\| [\Theta]_c + \|N_2\| [\Theta]_c + \\ & \text{nrec}(\|M\| [\Theta]_p, \|N_1\| [\Theta]_p, \|N_2\| [\Theta]_p^*)_c - (r_1 + r_2 + r_3 + r_4) \end{aligned}$$

$$- v \sqsubseteq_{\text{val}}^{C, f[\sigma]+g_1[\sigma]+g_2[\sigma]+r_1+r_2+r_3+r_4} \text{nrec}(\|M\| [\Theta]_p, \|N_1\| [\Theta]_p, \|N_2\| [\Theta]_p^*)_p$$

By IH,  $M[\theta] \sqsubseteq^{\mathbb{N}, f[\sigma]} \|M\| [\Theta]$ , so

$$- n_1 \leq \|M\| [\Theta]_c - r_1$$

$$- S(v_1) \sqsubseteq_{\text{val}}^{\mathbb{N}, f[\sigma]+r_1} \|M\| [\Theta]_p$$

Since  $S(v_1) \sqsubseteq_{\text{val}}^{\mathbb{N}, f[\sigma]+r_1} \|M\| [\Theta]_p$ , there is an  $E$  such that  $v_1 \sqsubseteq_{\text{val}}^{\mathbb{N}, f[\sigma]+r_1} E$ , and  $S(E) \leq \|M\| [\Theta]_p$ .

By IH,  $N_1[\theta] \sqsubseteq^{1-\circ C, g_1[\sigma]} \|N_1\| [\Theta]$ , and so

$$\begin{aligned} - n_3 &\leq \|N_1\| [\Theta]_c - r_3 \\ - \lambda x. N'_1 &\sqsubseteq_{\text{val}}^{1-\circ C, g_1[\sigma]+r_3} \|N_1\| [\Theta]_p \end{aligned}$$

By IH,  $N_2[\theta] \sqsubseteq_{\text{val}}^{!_0^\infty (\mathbb{N} \otimes (1-\circ C) - \circ C), g_2[\sigma]} \|N_2\| [\Theta]$ , so by definition,

$$\begin{aligned} - n_2 &\leq \|N_2\| [\Theta]_c - r_2 \\ - \text{save}_0^\infty \lambda x. N'_2 &\sqsubseteq_{\text{val}}^{!_0^\infty (\mathbb{N} \otimes (1-\circ C) - \circ C), g_2[\sigma]+r_2} \|N_2\| [\Theta]_p \end{aligned}$$

and so there is a  $d \geq 0$  so that  $\lambda x. N'_2 \sqsubseteq_{\text{val}}^{\mathbb{N} \otimes (1-\circ C) - \circ C, d} \|N_2\| [\Theta]_p$ , and  $\infty \cdot d \leq g_2[\sigma] + r_3$ . By the  $\mathbb{N}$ -recursor lemma,  $\text{nrec}(v_1, \lambda x. N'_1, \text{save}_0^\infty (\lambda x. N'_2)) \sqsubseteq_{\text{val}}^{C, g_1[\sigma]+r_2+\infty \cdot d} \text{nrec}(E, \|N_1\| [\Theta]_p, \|N_2\| [\Theta]_p^*)$ . Let  $E^* = (E, \text{nrec}(E, \|N_1\| [\Theta]_p, \|N_2\| [\Theta]_p^*))$ . Note that  $v_1 \sqsubseteq_{\text{val}}^{\mathbb{N}, f[\sigma]+r_1} \pi_1 E^*$  and

$$\lambda z. \text{nrec}(v_1, \lambda x. N'_1, \text{save}_0^\infty (\lambda x. N'_2)) \sqsubseteq_{\text{val}}^{1-\circ C, g_1[\sigma]+r_2+\infty \cdot d} \lambda z. \pi_2 E^*$$

, and so:

$$(v_1, \lambda z. \text{nrec}(v_1, \lambda x. N'_1, \text{save}_0^\infty (\lambda x. N'_2))) \sqsubseteq_{\text{val}}^{\mathbb{N} \otimes (1-\circ C), f[\sigma]+g_1[\sigma]+\infty \cdot d+r_1+r_2} (\pi_1 E^*, \lambda z. \pi_2 E^*)$$

Thus, because  $\lambda z. N'_2 \sqsubseteq_{\text{val}}^{\mathbb{N} \otimes (1-\circ C) - \circ C, d} \|N_2\| [\Theta]_p$ , and  $\infty \cdot d + d = \infty \cdot d$

$$N'_2[(v_1, \lambda z. \text{nrec}(v_1, \lambda x. N'_1, \text{save}_0^\infty (\lambda x. N'_2)))] \sqsubseteq_{\text{val}}^{C, f[\sigma]+g_1[\sigma]+\infty \cdot d+r_1+r_2} \|N_2\| [\Theta]_p (\pi_1 E^*, \lambda z. \pi_2 E^*)$$

and so:

$$\begin{aligned} - n_4 &\leq (\|N_2\| [\Theta]_p (\pi_1 E^*, \lambda z. \pi_2 E^*))_c - r_4 \\ - v &\sqsubseteq_{\text{val}}^{C, f[\sigma]+g_1[\sigma]+\infty \cdot d+r_1+r_2+r_4} (\|N_2\| [\Theta]_p (\pi_1 E^*, \lambda z. \pi_2 E^*))_p \\ \text{but, } \infty \cdot d &\leq g_2[\sigma] + r_3, \text{ and} \end{aligned}$$

$$\begin{aligned} \|N_2\| [\Theta]_p (\pi_1 E^*, \lambda z. \pi_2 E^*) &\leq (\lambda p. \|N_2\| [\Theta]_p (\pi_1 p, \lambda z. \pi_2 p)) E^* \\ &\leq \|N_2\| [\Theta]_p^* (E, \text{nrec}(E, \|N_1\| [\Theta]_p, \|N_2\| [\Theta]_p^*)) \\ &\leq \text{nrec}(S(E), \|N_1\| [\Theta]_p, \|N_2\| [\Theta]_p) \\ &\leq \text{nrec}(\|N_1\| [\Theta]_p, \|N_1\| [\Theta]_p, \|N_2\| [\Theta]_p) \end{aligned}$$

and so we are done by weakening and credit weakening.

(&-I) Suppose  $\Gamma \vdash_f \langle M, N \rangle : A \& B$ , and let  $\theta \sqsubseteq_{\text{sub}}^{\Gamma, \sigma} \Theta$ . We can invert to find that  $\Gamma \vdash_f M : A$  and  $\Gamma \vdash_f N : B$ . Since  $\langle M[\theta], N[\theta] \rangle \downarrow^{(0,0)} \langle M[\theta], N[\theta] \rangle$ , to show that  $\langle M[\theta], N[\theta] \rangle \sqsubseteq_{\text{val}}^{A \& B, f[\sigma]} (0, (\|M\| [\Theta], \|N\| [\Theta]))$  we must show that  $0 \leq 0$  (done!) and that  $\langle M[\theta], N[\theta] \rangle \sqsubseteq_{\text{val}}^{A \& B, f[\sigma]} (\|M\| [\Theta], \|N\| [\Theta])$ . For this, it suffices by weakening to show that  $M[\theta] \sqsubseteq_{\text{val}}^{A, f[\sigma]} \|M\| [\Theta]$  and  $N[\theta] \sqsubseteq_{\text{val}}^{B, f[\sigma]} \|N\| [\Theta]$ , which are precisely the inductive hypotheses.

(&-E) By symmetry, it suffices to present the  $\pi_1$  case. Suppose  $\Gamma \vdash_f \pi_1 M : A$ . By inversion,  $\Gamma \vdash_f M : A \& B$ . Let  $\theta \sqsubseteq_{\text{sub}}^{\Gamma, \sigma} \Theta$ . To show that  $\pi_1 M[\theta] \sqsubseteq_{\text{val}}^{A, f[\sigma]} \|M\| [\Theta]_c +_c \pi_1 (\|M\| [\Theta]_p)$ , assume  $\pi_1 M[\theta] \downarrow^{(n_1+n_2, r_1+r_2)} v$ . By inversion, it was by way of  $M[\theta] \downarrow^{(n_1, r_1)} \langle N_1, N_2 \rangle$  and  $N_1 \downarrow^{(n_2, r_2)} v$ . We must show that:

$$\begin{aligned} - n_1 + n_2 &\leq \|M\| [\Theta]_c + (\pi_1 \|M\| [\Theta]_p)_c - (r_1 + r_2) \\ - v &\sqsubseteq_{\text{val}}^{A, f[\sigma]+r_1+r_2} (\pi_1 \|M\| [\Theta]_p)_p \end{aligned}$$

By IH,  $M[\theta] \sqsubseteq_{\text{val}}^{A \& B, f[\sigma]} \|M\| [\Theta]$ , so

- $n_1 \leq \|M\| [\Theta]_c - r_1$
- $\langle N_1, N_2 \rangle \sqsubseteq_{\text{val}}^{A \& B, f[\sigma] + r_1} \|M\| [\Theta]_p$

where the second condition means, in particular, that  $N_1 \sqsubseteq_{\text{val}}^{A, f[\sigma] + r_1} \pi_1 \|M\| [\Theta]_p$ . So, since  $N_1 \downarrow^{(n_2, r_2)} v$ ,

- $n_2 \leq (\pi_1 \|M\| [\Theta]_p)_c - r_2$
  - $v \sqsubseteq_{\text{val}}^{A, f[\sigma] + r_1 + r_2} (\pi_1 \|M\| [\Theta]_p)_p$
- as required.

(var) Suppose  $\Gamma, x : A \vdash_{x+f} x : A$ . Let  $(\theta, v/x) \sqsubseteq_{\text{sub}}^{(\Gamma, x:A), (\sigma, x \mapsto a)} (\Theta, E/x)$ . We know that  $v \sqsubseteq_{\text{val}}^{A, a} E$ . We must show that  $v \sqsubseteq_{\text{val}}^{A, a+f[\sigma]} (0, E)$ . We know that  $v \downarrow^{(0,0)} v$ . Of course,  $0 \leq 0$ . Since  $f[\sigma] \geq 0$ , we are done by credit weakening.  $\square$

**THEOREM 4.1.** *For any posets  $A, B, C, G$  with  $\infty$  and  $\vee$ ,*

- (1)  $\text{snrec} \in \text{Hom}_{\text{Poset}} \left( (C^1)^G \times (C^{\mathbb{N} \times C})^G, C^{G \times \mathbb{N}} \right)$
- (2)  $\text{slrec} \in \text{Hom}_{\text{Poset}} \left( (C^1)^G \times (C^{A \times (\mathbb{N} \times C)})^G, C^{G \times \mathbb{N}} \right)$
- (3)  $\text{scaes} \in \text{Hom}_{\text{Poset}} \left( C^{G \times A} \times C^{G \times B}, C^{G \times (A+B)} \right)$

**PROOF.** Let  $A, B, C, G$  be posets. Note that these are not required to be in the image of  $\llbracket \cdot \rrbracket$ . For each case we must show two statements: the function is in fact monotonic, and that the functions in its image (an exponential poset) are themselves monotonic.

- (1) Suppose  $(f, g) \leq (f', g')$  as elements of  $(C^1)^G \times (C^{\mathbb{N} \times C})^G$ . To show that  $\text{snrec}(f, g) \leq \text{snrec}(f', g')$ , it suffices to show that for all  $\gamma, n$ , that  $\text{snrec}(f, g)(\gamma, n) \leq \text{snrec}(f', g')(\gamma, n)$ . Proceed by induction on  $n$ .

- $n = 0$ . By the definition of  $\text{snrec}$ , it suffices to show that  $f(\gamma)() \leq f'(\gamma)()$ , which is true since  $f \leq f'$ .
- $n + 1$  By definition of  $\text{snrec}$ , it suffices to show  $g(\gamma)(n, \text{snrec}(f, g)(\gamma, n)) \vee f(\gamma)() \leq g'(\gamma)(n, \text{snrec}(f', g')(\gamma, n)) \vee f'(\gamma)()$ . We have already shown that  $f(\gamma)() \leq f'(\gamma)()$ , so it remains to show  $g(\gamma)(n, \text{snrec}(f, g)(\gamma, n)) \leq g'(\gamma)(n, \text{snrec}(f', g')(\gamma, n))$ . Since  $g \leq g'$ ,  $g(\gamma) \leq g'(\gamma)$ . By reflexivity,  $n \leq n$ . By IH,  $\text{snrec}(f, g)(\gamma, n) \leq \text{snrec}(f', g')(\gamma, n)$ , and so we are done.

Now, let  $(f, g) \in (C^1)^G \times (C^{\mathbb{N} \times C})^G$ . We must show that if  $(\gamma, n) \leq (\gamma', n')$  in  $G \times \mathbb{N}$ , then  $\text{snrec}(f, g)(\gamma, n) \leq \text{snrec}(f, g)(\gamma', n')$ . Proceed by induction on  $n$ . We have three cases to consider.

- $n = n' = 0$ . By definition of  $\text{snrec}$ , it suffices to show that  $f(\gamma)() \leq f(\gamma')()$ , which is true since  $\gamma \leq \gamma'$ .
- $n = 0, n' + 1$ : By the definition of  $\text{snrec}$ , we must show that  $f(\gamma)() \leq g(\gamma')(n', \text{snrec}(f, g)(\gamma', n')) \vee f(\gamma')()$ , for which it suffices to show  $f(\gamma)() \leq f(\gamma')()$ , which we already argued was true.
- $n + 1, n' + 1$ . Expanding definitions again and simplifying, it suffices to show that  $g(\gamma)(n, \text{snrec}(f, g)(\gamma, n)) \leq g(\gamma')(n', \text{snrec}(f, g)(\gamma', n'))$ . Since  $g$  is monotonic,  $g(\gamma) \leq g(\gamma')$ . Since  $n + 1 \leq n' + 1, n \leq n'$ . By IH,  $\text{snrec}(f, g)(\gamma, n) \leq \text{snrec}(f, g)(\gamma', n')$ , and so  $g(\gamma)(n, \text{snrec}(f, g)(\gamma, n)) \leq g(\gamma')(n', \text{snrec}(f, g)(\gamma', n'))$ , as required.

- (2) Let  $(f, g) \leq (f', g') \in (C^1)^G \times (C^{A \times (\mathbb{N} \times C)})^G$ . We want to show that  $\text{slrec}(f, g) \leq \text{slrec}(f', g')$ . Fix  $\gamma \in G$ , we prove by induction on  $n$  that for all  $n \in \mathbb{N}$ ,  $\text{slrec}(f, g)(\gamma, n) \leq \text{slrec}(f', g')(\gamma, n)$ .

- $n = 0$ : expanding the definition of  $\text{slrec}$ , we must show that  $f(\gamma)() \leq f'(\gamma)()$ , which is true because  $f \leq f'$ .
- $n > 0$ . It suffices to show that  $g(\gamma)(\infty, (n, \text{slrec}(f, g)(\gamma, n))) \leq g'(\gamma)(\infty, (n, \text{slrec}(f', g')(\gamma, n)))$ . Since  $g \leq g'$ ,  $g(\gamma) \leq g'(\gamma)$ . Further,  $\infty \leq \infty$ ,  $n \leq n$ , and by IH,  $\text{slrec}(f, g)(\gamma, n) \leq \text{slrec}(f', g')(\gamma, n)$ , as required.

Now, let  $(f, g) \in (C^1)^G \times (C^{A \times (\mathbb{N} \times C)})^G$ . We must show that if  $(\gamma, n) \leq (\gamma', n')$ ,  $\text{slrec}(f, g)(\gamma, n) \leq \text{slrec}(f, g)(\gamma, n')$ . We again prove this by induction on  $n$ . There are three cases we must consider.

- $n = n' = 0$ . Immediate.
- $n = 0, n' > 0$ . Identical to the similar case for  $\text{snrec}$ .
- $n, n' > 0$ . To show that

$$g(\gamma)(\infty, (n, \text{snrec}(f, g)(\gamma, n))) \vee f(\gamma)() \leq g(\gamma')(\infty, (n', \text{snrec}(f, g)(\gamma', n))) \vee f(\gamma')()$$

it suffices to show that  $f(\gamma) \leq f(\gamma')$  (which is true because  $\gamma \leq \gamma'$  and  $f$  is monotonic) and  $g(\gamma)(\infty, (n, \text{snrec}(f, g)(\gamma, n))) \leq g(\gamma')(\infty, (n', \text{snrec}(f, g)(\gamma', n')))$ . Since  $n + 1 \leq n' + 1$ ,  $n \leq n'$ , and so the desired result follows from IH and the fact that  $g(\gamma)$

- (3) Let  $(f, g) \leq (f', g') \in C^{G \times A} \times C^{G \times B}$ . We must show that  $\text{sca}(f, g) \leq \text{sca}(f', g')$  in  $C^{G \times (A+B)}$ . Let  $(\gamma, x) \in G \times (A+B)$ . The two cases for  $x$  are symmetrical, so we consider when  $x = \text{inl } a$ . Then,

$$\begin{aligned} \text{sca}(f, g)(\gamma, \text{inl } a) &= f(\gamma, a) \vee g(\gamma, \infty) \\ &\leq f'(\gamma, a) \vee g(\gamma, \infty) \\ &= \text{sca}(f', g')(\gamma, \text{inl } a) \end{aligned}$$

as required.

Now, fix  $(f, g) \in C^{G \times A} \times C^{G \times B}$ . We must show that for all  $(\gamma, x) \leq (\gamma', y)$ ,  $\text{sca}(f, g)(\gamma, x) \leq \text{sca}(f, g)(\gamma', y)$ . We have two symmetric cases to consider, so we present the case where  $x = \text{inl } a$  and  $y = \text{inl } a'$ . Then,

$$\begin{aligned} \text{sca}(f, g)(\gamma, \text{inl } a) &= f(\gamma, a) \vee g(\gamma, \infty) \\ &\leq f(\gamma', a') \vee g(\gamma', \infty) \\ &= \text{sca}(f, g)(\gamma', \text{inl } a') \end{aligned}$$

as required.

□

**THEOREM 4.2 (COMPOSITIONALITY).** *If  $\Gamma, x : T_1 \vdash E : T_2$ , and  $\Gamma \vdash E' : T_1$ , then  $\llbracket \Gamma \vdash E[E'/x] : T_2 \rrbracket = (1_{\llbracket \Gamma \rrbracket}, \llbracket \Gamma \vdash E' : T_1 \rrbracket); \llbracket \Gamma, x : T_1 \vdash E : T_2 \rrbracket$*

**PROOF.** By induction on  $\Gamma, x : T_1 \vdash E : T_2$ .

- (nrec): Suppose  $\Gamma, x : T_1 \vdash \text{nrec}(E, E_1, E_2) : T_2$ . By inversion,  $\Gamma x : T_1 \vdash E : \mathbb{N}$ ,  $\Gamma, x : T_1 \vdash E_1 : 1 \rightarrow T_2$ , and  $\Gamma, x : T_1 \vdash E_2 : \mathbb{N} \times T_2 \rightarrow T_2$ . By IH,
  - $\llbracket \Gamma \vdash E[E'/x] : \mathbb{N} \rrbracket = (1_{\llbracket \Gamma \rrbracket}, \llbracket \Gamma \vdash E' : T_1 \rrbracket); \llbracket \Gamma, x : T_1 \vdash E : \mathbb{N} \rrbracket$
  - $\llbracket \Gamma \vdash E_1[E'/x] : 1 \rightarrow T_2 \rrbracket = (1_{\llbracket \Gamma \rrbracket}, \llbracket \Gamma \vdash E' : T_1 \rrbracket); \llbracket \Gamma, x : T_1 \vdash E_1 : 1 \rightarrow T_2 \rrbracket$
  - $\llbracket \Gamma \vdash E_2[E'/x] : \mathbb{N} \times T_2 \rightarrow T_2 \rrbracket = (1_{\llbracket \Gamma \rrbracket}, \llbracket \Gamma \vdash E' : T_1 \rrbracket); \llbracket \Gamma, x : T_1 \vdash E_2 : \mathbb{N} \times T_2 \rightarrow T_2 \rrbracket$

. For ease of notation, we let  $f = \llbracket \Gamma \vdash E' : T_1 \rrbracket$ ,  $g = \llbracket \Gamma, x : T_1 \vdash E : \mathbb{N} \rrbracket$ ,  $h_1 = \llbracket \Gamma, x : T_1 \vdash E_1 : 1 \rightarrow T_2 \rrbracket$ , and  $h_2 = \llbracket \Gamma, x : T_1 \vdash E_2 : \mathbb{N} \times T_2 \rightarrow T_2 \rrbracket$ . Then, we compute:

$$\begin{aligned} & \llbracket \Gamma \vdash (\text{nrec}(E, E_1, E_2)) [E'/x] : T_2 \rrbracket \\ &= \llbracket \Gamma \vdash \text{nrec}(E[E'/x], E_1[E'/x], E_2[E'/x]) \rrbracket \\ &= (1_{\llbracket \Gamma \rrbracket}, \llbracket \Gamma \vdash E[E'/x] : \mathbb{N} \rrbracket); \text{snrec}(\llbracket \Gamma \vdash E_1[E'/x] : 1 \rightarrow T_2 \rrbracket, \llbracket \Gamma \vdash E_2[E'/x] : \mathbb{N} \times T_2 \rightarrow T_2 \rrbracket) \\ &= (1_{\llbracket \Gamma \rrbracket}, (1_{\llbracket \Gamma \rrbracket}, f); g); \text{snrec}((1_{\llbracket \Gamma \rrbracket}, f); h_1, (1_{\llbracket \Gamma \rrbracket}, f); h_2) \end{aligned}$$

It remains to show that

$$(1_{\llbracket \Gamma \rrbracket}, (1_{\llbracket \Gamma \rrbracket}, f); g); \text{snrec}((1_{\llbracket \Gamma \rrbracket}, f); h_1, (1_{\llbracket \Gamma \rrbracket}, f); h_2) = (1_{\llbracket \Gamma \rrbracket}, f); (1_{\llbracket \Gamma, x : T_1 \rrbracket}, g); \text{snrec}(h_1, h_2)$$

Let  $\gamma \in \llbracket \Gamma \rrbracket$ . Applying the left hand side to  $\gamma$ , we get

$$\text{snrec}((1_{\llbracket \Gamma \rrbracket}, f); h_1, (1_{\llbracket \Gamma \rrbracket}, f); h_2)(\gamma, g(\gamma, f(\gamma)))$$

and on the right:

$$\text{snrec}(h_1, h_2)((\gamma, f(\gamma)), g(\gamma, f(\gamma)))$$

Letting  $\gamma' = (\gamma, f(\gamma))$ , we must show that  $\text{snrec}((1_{\llbracket \Gamma \rrbracket}, f); h_1, (1_{\llbracket \Gamma \rrbracket}, f); h_2)(\gamma, g(\gamma')) = \text{snrec}(h_1, h_2)(\gamma', g)$ . We proceed by induction on  $n = g(\gamma')$ .

–  $n = 0$ .

$$\begin{aligned} \text{snrec}((1_{\llbracket \Gamma \rrbracket}, f); h_1, (1_{\llbracket \Gamma \rrbracket}, f); h_2)(\gamma, 0) &= ((1_{\llbracket \Gamma \rrbracket}, f); h_1)(\gamma)() \\ &= h_1(\gamma, f(\gamma))() \\ &= h_1(\gamma')() \\ \text{snrec}(h_1, h_2)(\gamma', 0) &= h_1(\gamma')() \end{aligned}$$

as required.

–  $n + 1$ :

$$\begin{aligned} & \text{snrec}((1_{\llbracket \Gamma \rrbracket}, f); h_1, (1_{\llbracket \Gamma \rrbracket}, f); h_2)(\gamma, n + 1) \\ &= ((1_{\llbracket \Gamma \rrbracket}, f); h_1)(\gamma)(n, \text{snrec}((1_{\llbracket \Gamma \rrbracket}, f); h_1, (1_{\llbracket \Gamma \rrbracket}, f); h_2)(\gamma, n)) \vee h_1(\gamma')() \\ &= h_1(\gamma')(n, \text{snrec}(h_1, h_2)(\gamma', n)) \vee h_1(\gamma')() \\ &= \text{snrec}(h_1, h_2)(\gamma', n + 1) \end{aligned}$$

- (lrec): Suppose  $\Gamma, x : T_1 \vdash \text{lrec}(E', E_1, E_2) : T_2$ , and  $\Gamma \vdash E : T_1$ . By inversion,  $\Gamma, x : T_1 \vdash E : \text{List}(T)$ ,  $\Gamma, x : T_1 \vdash E_1 : 1 \rightarrow T_2$ , and  $\Gamma, x : T_1 \vdash E_2 : T \times (\text{List}(T) \times T_2) \rightarrow T_2$ . By IH, we have that:

$$\begin{aligned} & \llbracket \Gamma \vdash E'[E/x] : \text{List}(T) \rrbracket = (1_{\llbracket \Gamma \rrbracket}, \llbracket \Gamma \vdash E : T_1 \rrbracket); \llbracket \Gamma, x : T_1 \vdash E : \text{List}(T) \rrbracket \\ & \llbracket \Gamma \vdash E_1[E/x] : 1 \rightarrow T_2 \rrbracket = (1_{\llbracket \Gamma \rrbracket}, \llbracket \Gamma \vdash E : T_1 \rrbracket); \llbracket \Gamma, x : T_1 \vdash E_1 : 1 \rightarrow T_2 \rrbracket \\ & \llbracket \Gamma \vdash E_2[E/x] : T \times (\text{List}(T) \times T_2) \rightarrow T_2 \rrbracket = (1_{\llbracket \Gamma \rrbracket}, \llbracket \Gamma \vdash E : T_1 \rrbracket); \llbracket \Gamma, x : T_1 \vdash E_2 : T \times (\text{List}(T) \times T_2) \rightarrow T_2 \rrbracket \end{aligned}$$

Let  $f = \llbracket \Gamma \vdash E : T_1 \rrbracket$ ,  $g = \llbracket \Gamma, x : T_1 \vdash E' : \text{List}(T) \rrbracket$ ,  $h_1 = \llbracket \Gamma, x : T_1 \vdash E_1 : 1 \rightarrow T_2 \rrbracket$ , and  $h_2 = \llbracket \Gamma, x : T_1 \vdash E_2 : T \times (\text{List}(T) \times T_2) \rightarrow T_2 \rrbracket$

We must show that

$$(1_{\llbracket \Gamma \rrbracket}, f); (1_{\llbracket \Gamma \rrbracket \times \llbracket T_1 \rrbracket}, g); \text{slrec}(h_1, h_2) = (1_{\llbracket \Gamma \rrbracket}, (1_{\llbracket \Gamma \rrbracket}, f); g); \text{slrec}((1_{\llbracket \Gamma \rrbracket}, f); h_1, (1_{\llbracket \Gamma \rrbracket}, f); h_2)$$

Let  $\gamma \in \llbracket \Gamma \rrbracket$ , and let  $\gamma' = (\gamma, f(\gamma))$ . We must then show that

$$\text{slrec}(h_1, h_2)(\gamma', g(\gamma')) = \text{slrec}((1_{\llbracket \Gamma \rrbracket}, f); h_1, (1_{\llbracket \Gamma \rrbracket}, f); h_2)(\gamma, g(\gamma'))$$

We proceed by induction on  $n = g(\gamma')$ .



- ( $n = 0$ ): The LHS is  $\text{slrec}(h_1, h_2)(\gamma', 0) = h_1(\gamma')()$ , and the RHS is

$$\text{slrec}((1_{\llbracket \gamma \rrbracket}, f); h_1, (1_{\llbracket \gamma \rrbracket}, f); h_2)(\gamma, 0) = h_1(\gamma, f(\gamma))() = h_1(\gamma')()$$

- ( $n > 0$ ): The LHS is:

$$\begin{aligned} & \text{slrec}(h_1, h_2)(\gamma', n+1) \\ &= h_2(\gamma')(\infty, (n, \text{slrec}(h_1, h_2)(\gamma', n))) \vee h_1(\gamma')() \end{aligned}$$

and the RHS (applying the IH in the 2nd step) is

$$\begin{aligned} & \text{slrec}((1_{\llbracket \Gamma \rrbracket}, f); h_1, (1_{\llbracket \Gamma \rrbracket}, f); h_2)(\gamma, n+1) \\ &= h_2(\gamma')(\infty, (n, \text{slrec}((1_{\llbracket \Gamma \rrbracket}, f); h_1, (1_{\llbracket \Gamma \rrbracket}, f); h_2)(\gamma, n))) \vee h_1(\gamma')() \\ &= h_2(\gamma')(\gamma, (n, \text{slrec}(h_1, h_2)(\gamma', n))) \vee h_1(\gamma')() \end{aligned}$$

as required.  $\square$

PROOF. By induction on  $\Gamma \vdash E : T$ .

- (nrec): Let  $\Gamma \vdash \text{nrec}(E, E_1, E_2) : T$ . By inversion,  $\Gamma \vdash E : \mathbb{N}$ ,  $\Gamma \vdash E_1 : 1 \rightarrow T$ , and  $\Gamma \vdash E_2 : \mathbb{N} \times C \rightarrow C$ . By IH,  $\llbracket \Gamma \vdash E : \mathbb{N} \rrbracket \in \text{Hom}(\llbracket \Gamma \rrbracket, \mathbb{N})$ . Then,  $(1_\Gamma, \llbracket \Gamma \vdash E : \mathbb{N} \rrbracket) \in \text{Hom}(\llbracket \Gamma \rrbracket, \llbracket \Gamma \rrbracket \times \mathbb{N})$ . By IH,  $\llbracket \Gamma \vdash E_1 : 1 \rightarrow T \rrbracket \in \text{Hom}(\llbracket \Gamma \rrbracket, \llbracket T \rrbracket^1)$  and  $\llbracket \Gamma \vdash E_2 : \mathbb{N} \times T \rightarrow T \rrbracket \in \text{Hom}(\llbracket \Gamma \rrbracket, \llbracket T \rrbracket^{\mathbb{N} \times \llbracket T \rrbracket})$ . Then, by Theorem 4.1 and composition,  $(1_\Gamma, \llbracket \Gamma \vdash E : \mathbb{N} \rrbracket); \text{snrec}(\llbracket \Gamma \vdash E_1 : 1 \rightarrow T \rrbracket, \llbracket \Gamma \vdash E_2 : \mathbb{N} \times T \rightarrow T \rrbracket) \in \text{Hom}(\llbracket \Gamma \rrbracket, \llbracket T \rrbracket)$ , as required.
- (lrec): Let  $\Gamma \vdash \text{lrec}(E, E_1, E_2) : T$ . By inversion,  $\Gamma \vdash E : \text{List}(T')$ ,  $\Gamma \vdash E_2 : 1 \rightarrow T$ , and  $\Gamma \vdash E_2 : T' \times (\text{List}(T') \times T) \rightarrow T$ . Applying the IH to all of these premises, we have that  $\llbracket \Gamma \vdash E : \text{List}(T') \rrbracket \in \text{Hom}(\llbracket \Gamma \rrbracket, \mathbb{N})$ ,  $\llbracket \Gamma \vdash E_2 : 1 \rightarrow T \rrbracket \in \text{Hom}(\llbracket \Gamma \rrbracket, \llbracket T \rrbracket^1)$ , and  $\llbracket \Gamma \vdash E_2 : T' \times (\text{List}(T') \times T) \rightarrow T \rrbracket \in \text{Hom}(\llbracket \Gamma \rrbracket, \llbracket T \rrbracket^{\llbracket T' \rrbracket \times (\mathbb{N} \times \llbracket T \rrbracket)})$ . By Theorem 4.1 and composition,  $(1_{\llbracket \Gamma \rrbracket}, \llbracket \Gamma \vdash E : \text{List}(T') \rrbracket); \text{slrec}(\llbracket \Gamma \vdash E_2 : 1 \rightarrow T \rrbracket, \llbracket \Gamma \vdash E_2 : T' \times (\text{List}(T') \times T) \rightarrow T \rrbracket) \in \text{Hom}(\llbracket \Gamma \rrbracket, \llbracket T \rrbracket)$  as required.  $\square$

**THEOREM 4.4 (SOUNDNESS (INEQUALITY)).** *If  $\Gamma \vdash E \leq E'$ , then for all  $\gamma \in \llbracket \Gamma \rrbracket$ ,  $\llbracket \Gamma \vdash E : T \rrbracket(\gamma) \leq \llbracket \Gamma \vdash E' : T \rrbracket(\gamma)$*

PROOF. By induction on  $\Gamma \vdash E \leq E'$ . The new cases (snrec and slrec) follow easily from the definitions.  $\square$

**THEOREM 5.1 (SUBSTITUTION).**

- If  $\Delta \vdash c$  credit and  $\Delta, \alpha \vdash c'$  credit, then  $\Delta \vdash c'[c/\alpha]$  credit
- If  $\Delta \vdash c$  credit and  $\Delta, \alpha \mid \Gamma \vdash_f M : A$ , then  $\Delta \mid \Gamma[c/\alpha] \vdash_{f[c/\alpha]} M[c/\alpha] : A[c/\alpha]$

PROOF. By induction on  $\Delta, \alpha \vdash c'$  credit and  $\Delta, \alpha \mid \Gamma \vdash_f M : A$ , respectively.  $\square$

**THEOREM 5.2 (PRESERVATION).** *If  $\cdot \vdash_a M : A$  and  $M \downarrow^{(n,r)} v$ , then  $a + r \geq 0$  and  $\cdot \vdash_{a+r} v : A$ .*

PROOF. The cases for all pre-existing rules are identical– the only new cases are for pack, unpack, and trec. We present only the final case of trec, as it is the most illustrative.

- (pack): Suppose that  $\cdot \vdash_a \text{pack}_{\alpha=\ell} M : \exists \alpha$  and  $\text{pack}_{\alpha=\ell} M \downarrow^{(n,r)} \text{pack}_{\alpha=\ell} v$  by way of  $\cdot \vdash_a M : A[\ell/\alpha]$  and  $M \downarrow^{(n,r)} v$ . By IH,  $\cdot \vdash_{a+r} v : A[\ell/\alpha]$  and  $a + r \geq 0$ . By the rule for pack,  $\cdot \vdash_{a+r} \text{pack}_{\alpha=\ell} v : \exists \alpha. A$ , as required.

- (unpack): Suppose that  $\cdot \vdash_{a+b} \text{unpack } (\alpha, x) = M \text{ in } N : C$  by way of  $\cdot \vdash_a M : \exists \alpha. A$  and  $\alpha | x : A \vdash_{b+x} N : C$  with  $\Delta \vdash C$ , and that  $\text{unpack } (\alpha, x) = M \text{ in } N \downarrow^{(n_1+n_2, r_1+r_2)} v$  by way of  $M \downarrow^{(n_1, r_1)} \text{pack}_{\alpha=\ell} v_1$  and  $N[\ell/\alpha, v_1/x] \downarrow^{(n_2, r_2)} v$ . By IH,  $\cdot \vdash_{a+r_1} v_1 : A[\ell/\alpha]$ . By credit variable substitution,  $\cdot | x : A[\ell/\alpha] \vdash_{b+x} N[\ell/\alpha] : C$ . By substitution,  $\cdot \vdash_{b+a+r_1} N[\ell/\alpha, v_1/x] LC$ . By IH,  $\cdot \vdash_{a+b+r_1+r_2} v : C$  and  $a + b + r_1 + r_2 \geq 0$  as required.
- (trec): Suppose:

$$\frac{\begin{array}{l} \cdot \vdash_f M : \text{tree}(A) \\ \cdot \vdash_{b_1} N_1 : !_0^\infty (1 \multimap C) \\ \cdot \vdash_{b_2} N_2 : !_0^\infty (A \multimap C) \\ \cdot \vdash_{b_3} N_3 : !_0^\infty (A \otimes \mathbb{N} \otimes A \otimes N \otimes (\text{tree}(A) \& C)^2 \multimap C) \\ \cdot \vdash_{b_4} N_4 : !_0^\infty (A \otimes \mathbb{N} \otimes A \otimes N \otimes (\text{tree}(A) \& C)^2 \multimap C) \\ \cdot \vdash_{b_5} N_5 : !_0^\infty (A \otimes \mathbb{N} \otimes A \otimes N \otimes A \otimes N(\text{tree}(A) \& C)^4 \multimap C) \end{array}}{\cdot \vdash_{a+\sum b_i} \text{trec}(M, N_1, N_2, N_3, N_4, N_5)}$$

and

$$\frac{\begin{array}{l} M \downarrow^{(n_0, r_0)} N(v_1, n_1, N(v_2, n_2, t_{00}, t_{01}), N(v_3, n_7, t_{10}, t_{11})) \\ N_i \downarrow^{(n_i, r_i)} \text{save}_0^\infty v'_i \quad (1 \leq i \leq 4) \\ N_5 \downarrow^{(n_5, r_5)} \text{save}_0^\infty (\lambda x. N'_5) \\ N'_5[(v_1, n_1, v_2, n_2, v_3, n_3, \langle t_{00}, \text{trec}(t_{00}, \text{save}_0^\infty v'_1, \dots, \text{save}_0^\infty (\lambda x. N'_5)), \dots \rangle / x)] \end{array}}{\text{trec}(M, N_1, N_2, N_3, N_4, N_5) \downarrow^{(\sum n_i, \sum r_i)} v}$$

By IH,  $\cdot \vdash_{a+r_0} N(\dots)$ . Hence, there are  $d_1, \dots, d_n$ , all non-negative, so that  $\sum d_i = a + r_0$ , and  $\cdot \vdash_{d_i} w_i : A_i$  where  $w_i$  is the  $i$ th value in the value which  $M$  evaluates to (in particular,  $\cdot \vdash_{d_1} v_1 : A$ , and  $\cdot \vdash_{d_6} t_{00} : \text{tree}(A)$ ). Again by IH, there are  $c_1, \dots, c_5$  so that  $\infty c_i \leq b_i + r_i$ , with  $\cdot \vdash_{c_i} v'_i$ . Thus,  $\cdot \vdash_{d_6+\sum c_i} \langle t_{00}, \text{trec}(t_{00}, \text{save}_0^\infty v'_1, \dots) \rangle$ , and similarly for the rest of the subtrees. This immediately implies

$$\cdot \vdash_{\sum d_i + 4 \sum c_i} (v_1, n_1, v_2, n_2, v_3, n_3, \langle t_{00}, \text{trec}(t_{00}, \text{save}_0^\infty v'_1, \dots) \rangle, \dots) : (A \otimes \mathbb{N})^3 \otimes (\text{tree}(A) \& C)^4$$

then by substitution

$$\cdot \vdash_{\sum d_i + c_5 + 4 \sum c_i} N'_5[(v_1, n_1, v_2, n_2, v_3, n_3, \langle t_{00}, \text{trec}(t_{00}, \text{save}_0^\infty v'_1, \dots) \rangle, \dots) / x] : C$$

The result follows immediately by weakening ( $\infty c_i \leq b_i + r_i$ ) and IH.  $\square$

**THEOREM 5.3 (EXTRACTION PRESERVES TYPES).** *If  $\Delta | \Gamma \vdash_f M : A$ , then  $\langle\langle \Delta \rangle\rangle, \langle\langle \Gamma \rangle\rangle \vdash \|M\| : \|A\|$*

**PROOF.** By induction on  $\Delta | \Gamma \vdash_f M : A$ .  $\square$

**THEOREM 5.4 (BOUNDING THEOREM).** *If  $\Delta | \Gamma \vdash_f M : A$ , then  $M \sqsubseteq^A \|M\|$*

**PROOF.**

- (pack): Suppose  $\Delta | \Gamma \vdash_f \text{pack}_{\alpha=c} M : \exists \alpha. A$  by way of  $\Delta | \Gamma \vdash_f M : A[c/\alpha]$ . Let  $\omega \sqsubseteq_{\text{credit}}^\Delta \Omega$  and  $\theta \sqsubseteq_{\text{sub}}^{\Gamma[\omega], \sigma} \Theta$ . We must show that  $\text{pack}_{\alpha=c[\omega]} M[\omega, \theta] \sqsubseteq^{\exists \alpha. A[\omega], f[\omega, \sigma]} (\|M\|_c [\Omega, \Theta], (c[\Omega], \|M\|_p [\Omega, \Theta]))$ . Suppose  $\text{pack}_{\alpha=c[\omega]} M[\omega, \theta] \downarrow^{(n, r)} \text{pack}_{\alpha=c[\omega]} v$  by way of  $M[\omega, \theta] \downarrow^{(n, r)} v$ . It suffices to show
  - $n + r \leq \|M\|_c [\Omega, \Theta]$
  - $\text{pack}_{\alpha=c[\omega]} v \sqsubseteq_{\text{val}}^{\exists \alpha. A[\omega], f[\omega, \sigma] + r} (c[\Omega], \|M\|_p [\Omega, \Theta])$
 The second item is equivalent to proving that  $c[\omega] \leq c[\Omega]$  (which is true because credit terms are monotone), and that  $v \sqsubseteq_{\text{val}}^{A[c/\alpha, \omega], f[\omega, \sigma] + r} \|M\|_p [\Omega, \Theta]$ , which follows immediately by IH.

- (unpack): Suppose that  $\Delta|\Gamma \vdash_{f+g} \text{unpack } (\alpha, x) = M \text{ in } N : C$  by way of  $\Delta|\Gamma \vdash_f M : \exists\alpha.A$  and  $\Delta, \alpha|\Gamma, x : A \vdash_{g+x} N : C$  with  $\alpha$  not free in  $C$ . Let  $\omega \sqsubseteq_{\text{credit}}^{\Delta} \Omega$  and  $\theta \sqsubseteq_{\text{sub}}^{\Gamma[\omega], \sigma} \Theta$ . Suppose  $\text{unpack } (\alpha, x) = M[\omega, \theta]$  in  $N[\omega, \theta] \downarrow^{(n_1+n_2, r_1+r_2)} v$  by way of  $M[\omega, \theta] \downarrow^{(n_1, r_1)} \text{pack}_{\alpha=\ell} v_1$  and  $N[\omega, \theta, \ell/\alpha, v_1/x] \downarrow^{(n_2, r_2)} v$ . It suffices to show that
  - $n_1 + n_2 + r_1 + r_2 \leq \|M\|_c [\Omega, \Theta] + \|N\|_c [\Omega, \Theta, \pi_1 \|M\|_p [\Omega, \Theta]\alpha, \pi_2 \|M\|_p [\Omega, \Theta]/x]$
  - $v \sqsubseteq_{\text{val}}^{C[\omega], f[\omega, \sigma] + g[\omega, \sigma] + r_1 + r_2} \|N\|_p [\Omega, \Theta, \pi_1 \|M\|_p [\Omega, \Theta]\alpha, \pi_2 \|M\|_p [\Omega, \Theta]/x]$
 By IH,  $M[\omega, \theta] \sqsubseteq_{\exists\alpha.A[\omega], f[\omega, \sigma]} \|M\| [\Omega, \Theta]$ , and so
  - $n_1 + r_1 \leq \|M\|_c [\Omega, \Theta]$
  - $\text{pack}_{\alpha=\ell} v_1 \sqsubseteq_{\text{val}}^{\exists\alpha.A[\omega], f[\omega, \sigma] + r_1} \|M\|_p [\Omega, \Theta]$
 which means that  $\ell \leq \pi_1 \|M\|_p [\Omega, \Theta]$  and that  $v_1 \sqsubseteq_{\text{val}}^{A[\omega, \ell/\alpha], f[\omega, \sigma] + r_1} \pi_2 \|M\|_p [\Omega, \Theta]$ . Hence,  $(\omega, \ell/\alpha) \sqsubseteq_{\text{credit}}^{\Delta, \alpha} (\Omega, \pi_1 \|M\|_p [\Omega, \Theta]/\alpha)$ , and  $(\theta, v_1/x) \sqsubseteq_{\text{sub}}^{\Gamma[\omega], x:A[\ell/\alpha], \sigma, x \mapsto f[\omega, \sigma] + r_1} (\Theta, \pi_2 \|M\|_p [\Omega, \Theta]/x)$ . Thus, by IH,

$$N[\omega, \theta, \ell/\alpha, v_1/x] \sqsubseteq_{C[\omega], g[\omega, \sigma] + f[\omega, \sigma] + r_1} \|N\| [\Omega, \Theta, \pi_1 \|M\|_p [\Omega, \Theta]/\alpha, \pi_2 \|M\|_p [\Omega, \Theta]/x]$$

By definition,

- $n_2 + r_2 \leq \|N\|_c [\Omega, \Theta, \pi_1 \|M\|_p [\Omega, \Theta]/\alpha, \pi_2 \|M\|_p [\Omega, \Theta]/x]$
  - $v \sqsubseteq_{\text{val}}^{C[\omega], f[\omega, \sigma] + g[\omega, \sigma] + r_1 + r_2} \|N\|_p [\Omega, \Theta, \pi_1 \|M\|_p [\Omega, \Theta]/\alpha, \pi_2 \|M\|_p [\Omega, \Theta]/x]$
- as required. □

$$\begin{array}{c}
\lambda^A \text{ rules:} \quad \frac{}{\Delta|\Gamma \vdash_f \text{Emp} : \text{tree}(A)} \\
\\
\frac{\Delta|\Gamma \vdash_{f_1} M_1 : A \quad \Delta|\Gamma \vdash_{f_2} M_2 : \mathbb{N} \quad \Delta|\Gamma \vdash_{g_1} N_1 : \text{tree}(A) \quad \Delta|\Gamma \vdash_{g_2} N_2 : \text{tree}(A)}{\Delta|\Gamma \vdash_{f_1+f_2+g_1+g_2} N(M_1, M_2, N_1, N_2) : \text{tree}(A)} \\
\\
\frac{\begin{array}{l}
\Delta|\Gamma \vdash_f M : \text{tree}(A) \\
\Delta|\Gamma \vdash_{g_1} N_1 : !_0^\infty(1 \multimap C) \\
\Delta|\Gamma \vdash_{g_2} N_2 : !_0^\infty(A \multimap C) \\
\Delta|\Gamma \vdash_{g_3} N_3 : !_0^\infty(A \otimes \mathbb{N} \otimes A \otimes N \otimes (\text{tree}(A) \& C)^2 \multimap C) \\
\Delta|\Gamma \vdash_{g_4} N_4 : !_0^\infty(A \otimes \mathbb{N} \otimes A \otimes N \otimes (\text{tree}(A) \& C)^2 \multimap C) \\
\Delta|\Gamma \vdash_{g_5} N_5 : !_0^\infty(A \otimes \mathbb{N} \otimes A \otimes N \otimes A \otimes N(\text{tree}(A) \& C)^4 \multimap C)
\end{array}}{\Delta|\Gamma \vdash_{f+\sum_{i=1}^5 g_i} \text{trec}(M, N_1, N_2, N_3, N_4, N_5) : C} \\
\\
\frac{\begin{array}{l}
M \downarrow^{(n_1, r_1)} N(v_1, n_1, N(v_2, n_2, t_{00}, t_{01}), N(v_3, n_3, t_{10}, t_{11})) \\
N_1 \downarrow^{(n_2, r_2)} v'_1 \\
N_2 \downarrow^{(n_3, r_3)} v'_2 \\
N_3 \downarrow^{(n_4, r_4)} v'_3 \\
N_4 \downarrow^{(n_5, r_5)} v'_4 \\
N_5 \downarrow^{(n_6, r_6)} \text{save}_0^\infty \lambda x. N'_5 \\
N'_5 \left[ \left( v_1, n_1, v_2, n_2, v_3, n_3, \langle t_{00}, \text{trec}(t_{00}, v'_1, v'_2, v'_3, v'_4, \text{save}_0^\infty \lambda x. N'_5) \rangle, \dots \right) / x \right] \downarrow^{(n_7, r_7)} v
\end{array}}{\text{trec}(M, N_1, N_2, N_3, N_4, N_5) \downarrow^{(\sum_{i=1}^7 n_i, \sum_{i=1}^7 r_i)} v} \\
\\
\lambda^C \text{ rules:} \quad \frac{}{\Gamma \vdash \text{Emp} : \text{tree}(T)} \quad \frac{\Gamma \vdash E_1 : T \quad \Gamma \vdash E_2 : \mathbb{N} \quad \Gamma \vdash E'_1 : \text{tree}(T) \quad \Gamma \vdash E'_2 : \text{tree}(T)}{\Gamma \vdash N(E_1, E_2, E'_1, E'_2) : \text{tree}(T)} \\
\\
\frac{\begin{array}{l}
\Gamma \vdash E : \text{tree}(T) \\
\Gamma \vdash E_1 : 1 \rightarrow T' \\
\Gamma \vdash E_2 : T \rightarrow T' \\
\Gamma \vdash E_3 : A \times \mathbb{N} \times A \times \mathbb{N} \times (\text{tree}(T) \times T')^2 \rightarrow T' \\
\Gamma \vdash E_4 : A \times \mathbb{N} \times A \times \mathbb{N} \times (\text{tree}(T) \times T')^2 \rightarrow T' \\
\Gamma \vdash E_5 : A \times \mathbb{N} \times A \times \mathbb{N} \times A \times \mathbb{N} \times (\text{tree}(T) \times T')^4 \rightarrow T'
\end{array}}{\Gamma \vdash \text{trec}(E, E_1, E_2, E_3, E_4, E_5) : T'} \\
\\
\| \text{trec}(M, N_1, N_2, N_3, N_4, N_5) \| = \left( \|M\|_c + \sum_{i=1}^5 \|N_i\|_c \right) +_c \\
\text{trec}(\|M\|_p, \|N_1\|_p, \|N_2\|_p, \lambda(x, n_1, y, n_2, (r_1, t_1), (r_2, t_2)). \|N_3\|_p(x_1, n_1, y, n_2, ((0, r_1), t_1), ((0, r_2), t_2)), \dots)
\end{array}$$

Fig. 15.  $\lambda^A$  and  $\lambda^C$  tree extension, and recurrence extraction